パソコンプログラマのためのクリエイティブマガジン
平成3年4月2日第三種郵便物認可　1992年5月1日発行(毎月1回18日)　第2巻第5号通巻17号

月刊プログラマーズ・ページ

Programmer's Page



1992 5 MAY

特集

Cの学び方・教え方

Cがわかる,うまくなる学習方法を探る

定価980円

付録:5"2HDフロッピーディスク

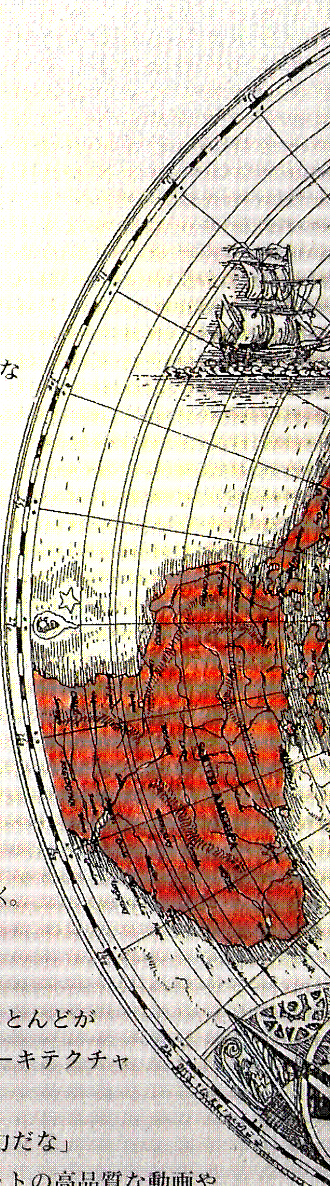
本誌掲載プログラム／綴じ込み付録テキストファイル（NEC PC-9800シリーズ MS-DOS Ver.5.0コマンド一覧）／クイズ・ランダムスクリーン解読ツール／多機能CDコマンド「FCD」／ディスク使用率表示ツール「gdf」／JPEGローダ(サンプル版)

SE
Shoeisha Co., Ltd.



世界標準物語　第三話

14本目の煙草



とある企業のオフィス。二人の男がテーブルをはさみ、真剣な顔で向き合っている。
テーブルの上の灰皿には、すでに12本の煙草の吸殻がある。

A部長「それじゃあキミはどうしても、AXがいいというんだね？」

X主任「はい、どうしてもAXがいいんです。今回だけは一歩も譲れません」

短い沈黙。A部長、13本目の煙草を取り出し、おもむろに火をつける。

A部長「しかし、うちの社内でAXを使っているのは、まだキミを含めて数人しかいないんだ。他のものが使いこなせるかね」

X主任「使いこなせるもなにも、操作自体は全く変わらないんです。違うのは、パソコン内部のアーキテクチャです。AXは、事実上の世界標準機であるIBMPC/ATと同じオープンアーキテクチャを採用しているので、世界に6万本以上といわれるIBMPC用ソフトがそのまま使えるんです。今、うちの社内で使っているパソコンは、日本では一応、標準機ということになってますが、閉ざされた、メーカー独自のアーキテクチャを採用しているので、そういうことはないんですよ」

A部長「IBMPCは、そんなに世界中で使われているのか？」

X主任、待ってましたとばかりに、一枚の世界地図をとりだす。

X主任「見てください。アメリカはもちろん、カナダ、メキシコ、ブラジル、イギリス、フランス、ドイツ、ソ連、中国…………。IBMPCとその互換機は日本以外のほとんどの国でトップシェアを占めているのが現実です」

X主任、つぎつぎに国名をあげながら、猛スピードでその国の位置を赤鉛筆で塗りつぶしていく。

A部長、思わず大きな溜息とともに煙を吐く。

A部長「うーん、これはすごい。日本はまさに世界の異端児ってわけだ」

X主任「MS-Windows、LAN、DTPソフトなど、現在我々が使っているソフトのほとんどが海外でIBMPC用として開発されたものです。日本語への移植も、AXなら基本アーキテクチャが同じなので即座に行えるというわけです」

A部長「GUI環境を実現するWindows3.0対応ソフトがすぐ使えるのは、確かに魅力だな」

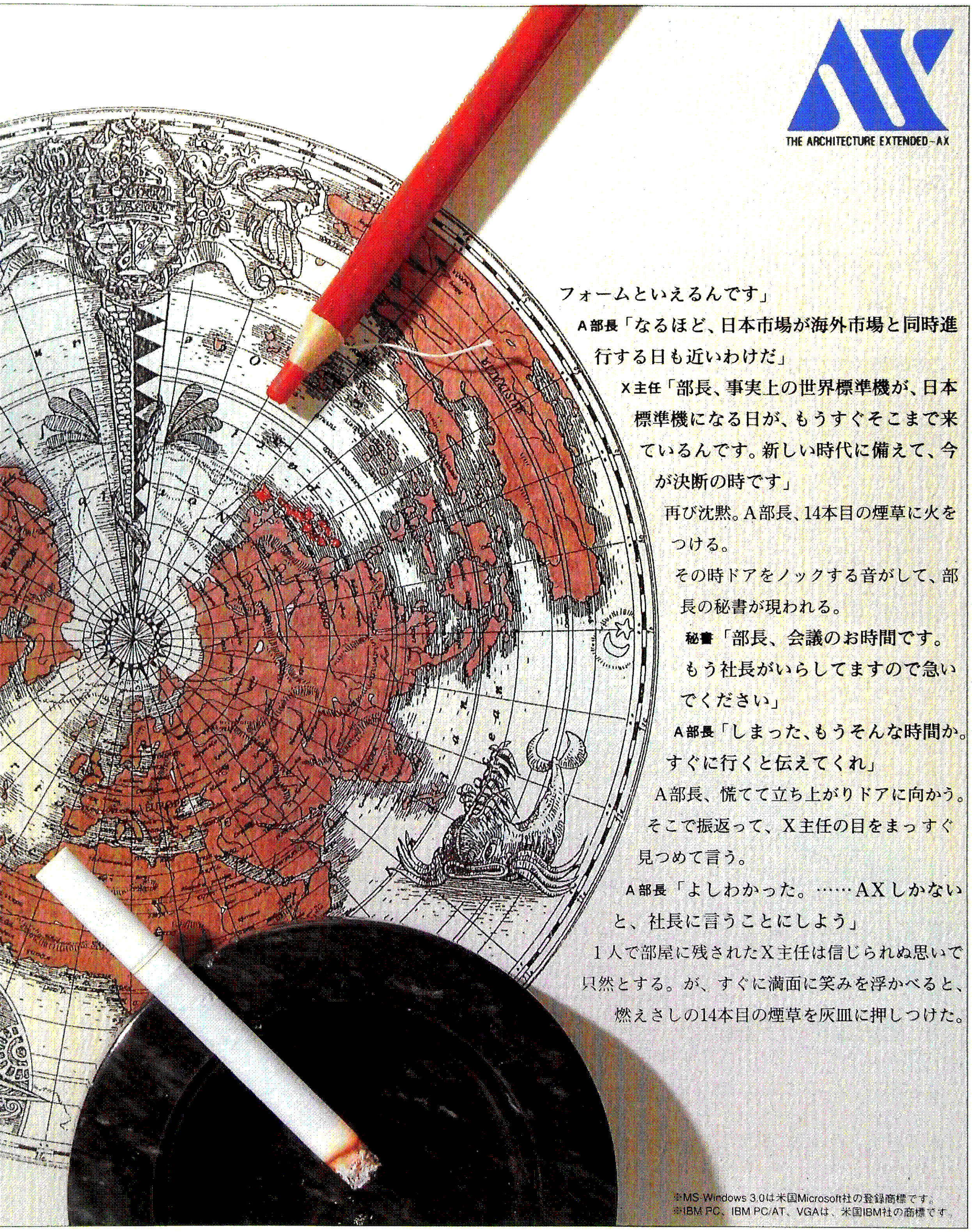
X主任「それにAXはVGAに対応していますからね。これらの画像は、640X480ドットの高品質な動画やアウトラインフォントを扱えるVGAモードでなくちゃ、世界のマルチメディアの流れにはついていけませんよ」

A部長、煙草を揉み消し、身を乗り出す。

X主任「オープンアーキテクチャのAXこそが、ネットワークコンピューティングを可能にする共通プラット

世界の標準機を、日本の標準機に。
我々が、パソコンの新しい時代を拓きます。

ACER　Canon
MITSUBISHI　NCR　OKI　olivetti　SANYO


THE ARCHITECTURE EXTENDED-AX
フォームといえるんです」
A部長「なるほど、日本市場が海外市場と同時進行する日も近いわけだ」
X主任「部長、事実上の世界標準機が、日本標準機になる日が、もうすぐそこまで来ているんです。新しい時代に備えて、今が決断の時です」
再び沈黙。A部長、14本目の煙草に火をつける。
その時ドアをノックする音がして、部長の秘書が現われる。
秘書「部長、会議のお時間です。もう社長がいらしてますので急いでください」
A部長「しまった、もうそんな時間か。すぐに行くと伝えてくれ」
A部長、慌てて立ち上がりドアに向かう。そこで振返って、X主任の目をまっすぐ見つめて言う。
A部長「よしわかった。……AXしかないと、社長に言うことにしよう」
1人で部屋に残されたX主任は信じられぬ思いで只然とする。が、すぐに満面に笑みを浮かべると、燃えさしの14本目の煙草を灰皿に押しつけた。
※MS-Windows 3.0は米国Microsoft社の登録商標です。
※IBM PC、IBM PC/AT、VGAは、米国IBM社の商標です。

Brief

Full Screen Editor 日本語版

全米のプログラマが認めた実力！
アメリカでは「ブリーフ」が
プログラミング用エディターとして
最も多く利用されています。

C言語で組めるマクロ！

Version UP!

V.3.0 いよいよ登場。
プログラミングエディター新世紀。



よりパワフルで、よりエレガントな プログラミングのために……。

●C言語で組めるマクロ 新機能

使い慣れているC言語でマクロが組めますから、習得が簡単でカスタマイズも思いのままです。

●マクロのスクリーン・デバッガ 新機能

ソースコードでデバックできます。もちろんブレークポイントやウォッチ・ドッグもセットできます。

●テンプレート機能〈構文の自動生成〉 新機能

C、Ada、Basic、COBOL、FORTRAN、MODULA2、Pascalで構文を自動生成します。

●300回ものアンドゥ

各ファイルごとに300までの操作を元にもどすことができますので、やり直しの面倒を気にせず、安心して編集できます。

●安全編集

Briefは、一定時間キーを操作しないこと最新の編集内容をファイルに自動保存します。だから編集中に電源が切れても安心です。

●その他の主な新機能

キー操作のファイルへのセーブとリプレイ、処理可能行数を拡張、リジューム機能、Ctrlキー・アサイン（オプション）、メニュー（オプション）etc.

アメリカでも日本でも高い評価

●プログラミング向きエディターを探している人は、もう探す必要はない。（Jerry Pournelle BYTE）
●ソフト開発には強力なツール、300回のアンドゥは大きな特徴。（日経バイト）
●既存のエディターに不満を覚えたら、これしかない。（Oh! FM）
●まるで夢のようなアンドゥだ……おもわず間違うのを期待してしまいそうだ。（The C Users Journal）

■主な機能

◆55ウィンドウ◆無限のファイル◆自動コンパイル◆自動エラー・ジャンプ◆自動インデント◆自動テンプレート◆バッファ書出 切換 削除 リスト表示◆ファイル名の指定 表示 変更◆バックアップ保存◆特殊文字入力◆2行を1行に連結◆ファイル単位の挿入◆増分検索◆各種マーク 始・終点の入換◆ブロックのカット コピー ペースト 印刷 保存 左右へのシフト◆対応する括弧へ移動◆C言語のコメント化 非コメント化◆DOSコマンドの実行◆キー操作記憶 実行◆入力ヒストリー◆英文テキスト整形◆タブのスペース挿入◆拡張子毎に異なった設定ができるタブ◆検索置換パターンの画面からの入力◆英字の大小文字変換 その他

定価 48,000円
（消費税は含まず）

■5機種対応パッケージ

（東芝、NEC、AX協議会、IBM、富士通）

●5 2DD、3.5 2DD デュアルメディア●MS-DOS 3.1以降に対応●英語DOS では使用できませんが、英語版 BRIEFもセット販売いたしております。●お急ぎの方、上記以外のメディア希望の方、英語BRIEF のセット販売希望の方はユーザー・サポート係までご連絡下さい。

日本システム株式会社

●お問合せは　東京都立川市曙町1-18-2一清ビル立川事業所Briefユーザー・サポート係
TEL0425-23-4469　FAX0425-27-6127

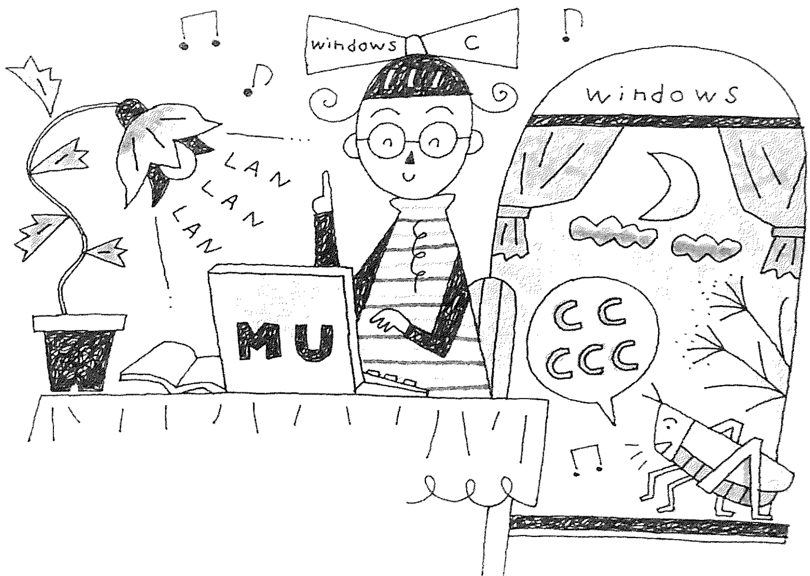
〈資料請求番号001〉

次代のアドバンテージ、最新の技術をあなたに!

Microsoft® University

独学より大学。

マルチメディアのプラットフォーム MS-Windows、高信頼性の MS OS/2。本格的なマルチタスクの実現、グラフィカルなユーザ・インタフェース(GUI)、LANマネージャによる高度なネットワーク処理など、技術者の創造力を刺激する環境を提供します。マイクロソフト・ユニバーシティは、その登竜門。ウインドウ環境、デバイスドライバの開発、ネットワーク処理など、アプリケーション・ソフト開発に欠かせない、ノウハウをレクチャします。入門から専門まで、充実したコースラインアップ。短時間で即開発に結びつく新しいコンピューティングスタイルを身につけて下さい。



Microsoft Windows Programming Environment
ウインドウズ・プログラミング・エンバイラメント

■期間:5日間 ■受講料:18万円(税別)

■MS Windows Ver3.0のアプリケーションを作成する方のためのコースです。Windows プログラミングのポイントを重点的に解説、実習中心のコースです。

Microsoft Windows Applications Programming
ウインドウズ・アプリケーションズ・プログラミング

■期間:5日間 ■受講料:18万円(税別)

■MS Windows Ver3.0対応のアプリケーションを開発するプログラマや設計者のためのコースです。ユーザインタフェースやアプリケーションなど設計ポイントがマスタできます。

Microsoft LAN Manager Programming Environment
LANマネージャ・プログラミング・エンバイラメント

■期間:4日間 ■受講料:15万円(税別)

■マイクロソフト LAN マネージャ(LM)用アプリケーションを開発する設計者のためのコースです。LMのアーキテクチャと特徴について解説します。

Microsoft LAN Manager Applications Programming
LANマネージャ・アプリケーションズ・プログラミング

■期間:5日間 ■受講料:18万円(税別)

■マイクロソフトLANマネージャ(LM)上で動作するアプリケーションの作成や移植するプログラマのためのコースです。

MS OS/2 Presentation Manager Programming Environment
プレゼンテーション・マネージャ・プログラミング・エンバイラメント

■期間:5日間 ■受講料:18万円(税別)

■MS OS/2プレゼンテーション・マネージャのアプリケーションを作成するための入門コースです。作成に必要となる基本事項を説明いたします。

MS OS/2 Presentation Manager Applications Programming
プレゼンテーション・マネージャ・アプリケーションズ・プログラミング

■期間:5日間 ■受講料:18万円(税別)

■MS OS/2 PMのアプリケーションを作成するためのステップアップコースです。経験者の方を対象に、より詳しく説明します。

MS OS/2 Programming Environment
プログラミング・エンバイラメント

■期間:4日間 ■受講料:15万円(税別)

■MS OS/2のアプリケーションを作成するために必要となる基本的技術を修得するコースです。本コースは、OS/2に取り組む上での基本コースとなります。

MS OS/2 Applications Programming
アプリケーションズ・プログラミング

■期間:5日間 ■受講料:18万円(税別)

■MS OS/2のAPIを使用し、マルチタスク、メモリ管理、プロセス間通信、ダイナミックリンクなどの機能を有効に活用する、実習を中心としたコースです。

MS OS/2 Device Drivers
デバイス・ドライバ

■期間:4日間 ■受講料:15万円(税別)

■MS OS/2のデバイス・ドライバの機能、構造を修得し、実際にデバイス・ドライバを作成するコースです。デバイスヘルパールーチンなども解説します。

C言語入門

■期間:2日間 ■受講料:6万円(税別)

■これからC言語を始めようとする方を対象に、C言語の基本的な文法からわかりやすく解説します 統合環境を持つQuick Cを使って理解を深めます

C言語応用

■期間:3日間 ■受講料:9万円(税別)

■関数、ポインタ、構造体についてプログラムを作成しながら詳しく説明します セミナは、最新のMicrosoft C Ver 6.0を使用して実施いたします

●お問い合せは
富士ソフトウエア株式会社 MU教育センタ
TEL.03-5461-8296 FAX.03-5461-8469

日本電気株式会社殿の御協力によりPC-9800シリーズ対応コースも好評開催中!!

Microsoft®
マイクロソフト株式会社
〒160 東京都新宿区西新宿7-5-25 Kビルディング

FSI 富士ソフトウエア株式会社
MU教育センタ
〒108 東京都港区港南2-12-27イケダヤ品川ビル

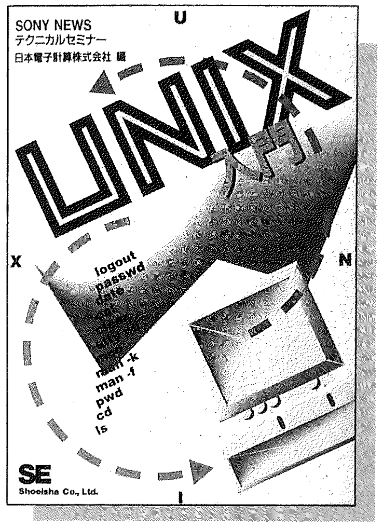
〈資料請求番号002〉

UNIX入門

―SONY NEWSテクニカルセミナー―

日本電子計算株式会社　編
A5判/237頁
定価1,800円

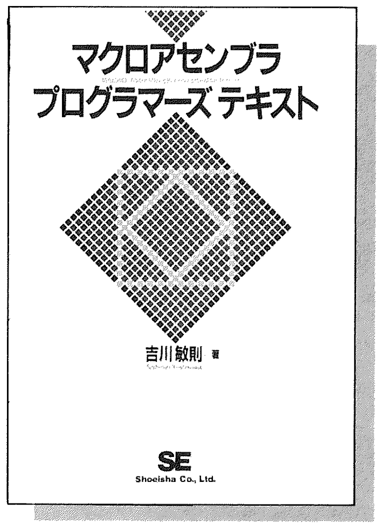
『ソニーのNEWSによる「UNIX入門」セミナーのテキストをベースとした入門書。UNIXのコマンド、エディタの使い方、日本語入力の方法、ネットワークの利用方法などを豊富な実例をまじえて解説。はじめてのワークステーションを利用するユーザ必携の書。』



マクロアセンブラ プログラマーズテキスト

吉川敏則　著
A5判/243頁
定価2,600円

『最低限必要なハードとソフトの知識から、データ演算の仕組みを踏まえた上で、アセンブラプログラミングの基本を丁寧に解説。アセンブリ言語に興味を持つ人や、これからアセンブリ言語でプログラムを作成する人に役立つ一冊。』



C文解釈

―「読む」ことからはじめるC言語―

豊国永健　著
B5判/183頁
定価2,000円

『C言語の文法書ではあまり取り上げられてこなかったわかりづらいポイントを、実際の応用プログラムを用いて解説。C言語の入門書とプログラムリスト中心の本との橋渡しになります。』



PC-PAGE

どれをとっても栄養満点!
おいしさいっぱいのパソコンブック

絶賛発売中



25 MS-DOS テキストデータ料理学

DOSのテキストデータを十二分に活用するためのガイドブック。

●定価1,500円・SE編集部編



24 MS-DOS5.0 全ファイル解説

98用MS-DOS5.0の全てのファイルの意味と利用法を解説。

●定価1,500円
SE編集部編



23 ハードディスクの達人

ビギナーでもハードディスクを120%活用できるマニュアル。

●定価1,500円
SE編集部編

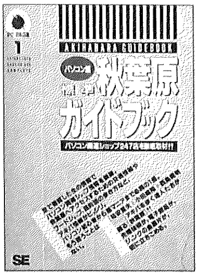


22 パソコンのお医者さん

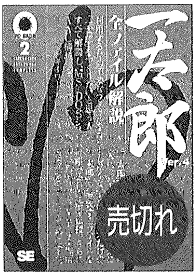
好評を博した「パソコン見栄講座」の著者による、パソコン版「家庭の医学」。

●定価1,500円
SE編集部編

SE Shoeisha Co., Ltd. 株式会社翔泳社 〒102東京都千代田区平河町2-4-14平河町KSビル TEL.03-3263-0447(代表)



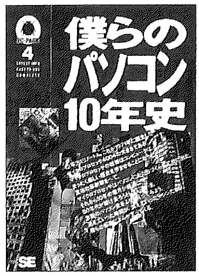
1 パソコン版標準秋葉原ガイドブック
定価1200円



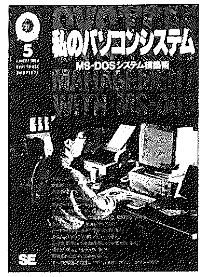
2 一太郎Ver.4全ファイル解説
定価1200円



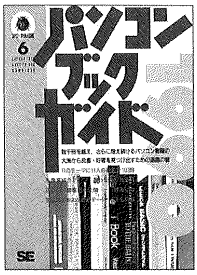
3 林檎百科
定価1200円



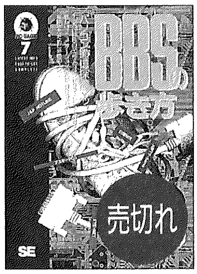
4 僕らのパソコン10年史
定価1200円



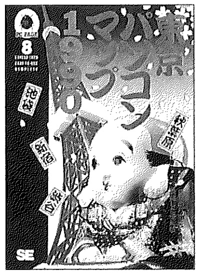
5 私のパソコンシステム
定価1200円



6 パソコンブックガイド1990
定価1200円



7 BBSの歩き方
定価1200円



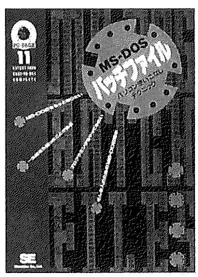
8 東京パソコンマップ1990
定価1200円



9 MS-DOS3.3全ファイル解説
定価1200円



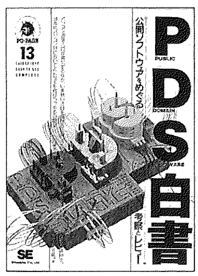
10 パソコンを思想する
定価1200円



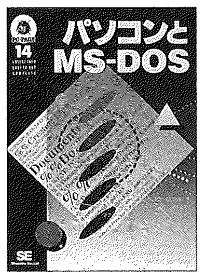
11 MS-DOSバッチファイル
定価1500円



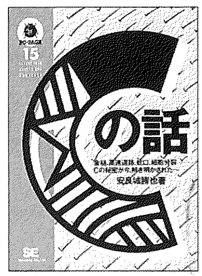
12 プロフェッショナルのパソコン利用術
定価1200円



13 PDS白書
定価1500円



14 パソコンとMS-DOS
定価1200円



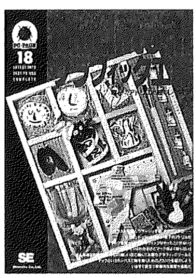
15 Cの話
定価1500円



16 パソコン見栄講座
定価1500円



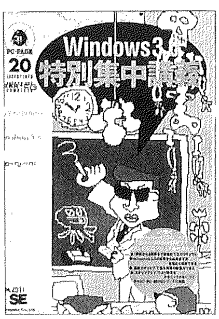
17 WINDOWSがやってきた
定価1200円



18 Macintoshグラフィック工房
定価1500円



19 日本語入力読本
定価1500円



20 Windows3.0特別集中講義
定価1500円



21 フリーソフトウェア派宣言
定価1500円

最新刊

NOW PRINTING

26
Mac派 DTP倶楽部
マックでミニメディアを作るための実践的解説書。

●定価1,600円・SE編集部編

※定価はすべて税込価格となっております。

翔泳社の本は全国どの書店でもお求めになれます。店頭にない場合は「至急」とお申し出の上ご注文いただければ10日位でお届けできます。

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

特集
Cの学び方・教え方
Cがわかる，うまくなる学習方法を探る

付録
5” 2HD フロッピーディスク

綴じ込み付録
NEC PC-9800シリーズ MS-DOS Ver.5.0コマンド一覧

表紙イラスト：日暮修一
表紙 AD：佐々木幹夫(SE)

広告索引　資料請求番号

人材募集広告

(掲載順)

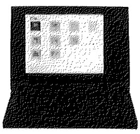
1992 MAY No.17

『ありがとう5周年』記念セール 5/1～5/31

IBM Net Ware 取扱開始！ NEWマシンぞくぞく大量入荷!!

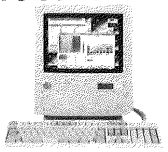
T・ZONEもおかげさまで5周年
ありがとうの気持を込めて！
●期間中5万円以上お買上げの方にすてきなオリジナルタオルをプレゼント！

PS/55 note



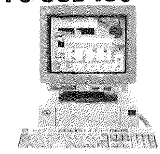
大容量80MBハードディスク内蔵
パワフルなDOS/Vノート
80386SX(16MHz)2MB FDD内蔵

PS 55Z 30U



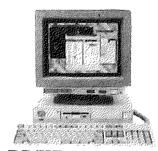
XGA採用パーソナルディスクトップ
80386SX(20MHz)2MB or 6MB
パソコンらしくないパソコン?
オシャレ

PS 55Z 10S



386SX搭載の
スタンダードPC
DOSV、VGA、ATバス採用
80386SX(20MHz)2MB or 6MB

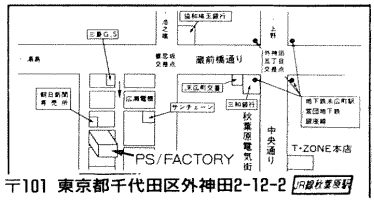
PS 55Z 10T



PS/55Z
最強パフォーマンスPC
XGA高解像度・大容量
80386(20MHz)2MB or 6MB

OPEN IBM PS/FACTORY お茶の水にオープン

Net Wareデモ中！ 何でもご相談下さい。



〒101 東京都千代田区外神田2-12-2 JR線秋葉原駅
TEL.03-3257-2750 FAX.03-3257-2751

ディスプレイディスパッチドライバー by C・F・Computing

今話題のS3ボードやTiGAボードに付属の英語ドライバーで日本語が使える!! 特価¥3,800

VGA DOS/Vの必須アイテムは早いVGAアダプタです。

ORCHID FAHRENHEIT 1280 1MB ¥89,800 ATI GRAPHICS VANTAGE 1MB ¥89,800

ATI GRAPHICS ULTRA 1MB ¥158,000
STB WIND/X(マウス付) 1MB ¥82,800
STB 1280 1MB ¥108,000
STB 1280 2MB ¥138,000
ET-4000 1MB ¥29,800

CREATIVELABS

マルチメディア対応のSOUND BLASTERは必需品です！ 特価提供中

SOUND BLASTER

これ1つでマルチメディアが楽しめます。
内蔵CD-ROM
SOUND BLASTER PRO
MIDI KIT
マルチメディアイクステンション他ソフト付
UPGRADE KIT ~~¥104,800~~

ステレオ22FM音源
CD-ROMインターフェイス他
SOUND BLASTER互換
Multimedia Sound Standard
PRO ~~¥57,800~~

11FM音源
MIDIジョイスティックパワーアンプマイク入力 各種ソフト付
V2.0 ~~¥34,800~~

PS/55等のマイクロチャンネル版です。
MCA ~~¥64,800~~

VGAモニター

DOS/V対応やWINDOWS用として
高解像大画面のモニターがほしいあなた、
何を選びますか？

NANAO T660i (ケーブル別)
20"0.31mm 30～78kHz 1280×1024 ~~¥598,000~~

三菱 XC-98V3
14"0.28mm 98/VGA 640×480 ¥78.000

TATUNG(マイコン搭載)
17" 0.28mm 29－65kHz 1280×1024
超特価 ¥180.000

Mitac 3060D

106キーセット 超特価 ¥258,000
101キーセット+DOS5.0 超特価¥258,000

共通仕様
80386SX(16MHz, 3.5"1.44MB×1、HD 40MB、スーパーVGA(512KB)、スーパーVGAモニター

DOS/V版 パソコン電子手帳くん ¥14,800 新登場

シャープの電子手帳とDOS/Vマシンの間で
データのやりとりができるようになりました。
対応している電子手帳
PA6000、6500、7000、7500、8500、8600シリーズ。

AST RESEARCH JAPAN K.K.

ASTから DOS/Vマシン

新登場

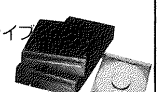


80386SX/20MHz
FDDモデル ~~¥308,000~~
40MBモデル ~~¥355,000~~

● 高速32ビットCPUを採用。
メインメモリ2MBを標準搭載
● 80387数値演算プロセッサの搭載が可能。
● IBM PC ATスロットを5基装備。

PCvision

マルチメディア対応高速CD-ROMドライブ



● 標準SCSI(最大7台まで同時接続可能)
● 高速アクセス(340ms)、64KBキャッシュ内蔵
● CD-ROM XA対応
● 市販オーディオディスク演奏可能

	ドライブのみ	AT I/F付	PS/2 I/F付
内蔵モデル	¥125,000	¥145,000	¥158,000
外付モデル	¥135,000	¥155,000	¥168,000

特価提供中

DOS/Vノートおすすめセット 限定

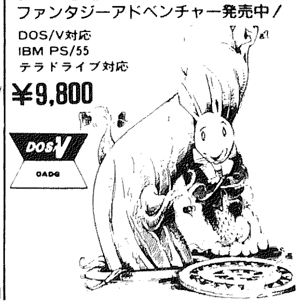
● CPU : 80386SX/20MHz
● RAM : 3MB
● FDD : 3.5"×1
● HDD : 40MB
● 外部VGA出力
● A4ファイルサイス

+ DOS J5.0/V + シリアルマウス + 一太郎 dash

何と！何と！の 超特価¥278,000

the Manhole (日本語版)

それはマンホールからはじまった……
ゲームを超えた
ファンタジーアドベンチャー発売中！
DOS/V対応
IBM PS/55
テラドライブ対応
¥9,800



80486/33MHz 超特価¥498,000

80486/33MHz
RAM 8MB(max 32MB)
128KBキャッシュ(max 256KB)
3.5"1.44MB×1/5"1.2MB×1
200MB IDE HDD
スーパーVGA ET-4000 W/1MB
2シリアル、1パラレル、1ゲーム
拡張スロット×8
ドライブベイ×6
ミニタワー、2ボタンマウス
106キーボード

80386/33MHz ノート made in TAIWAN

限定20台 超特価¥398,000 最強ノート

80386/33MHz
RAM 2MB(max 16MB)
32KBキャッシュ
3.5"1.44MB×1
40MB HDD
32階調VGAバックライト液晶
外部VGA出力
1シリアル、1パラレル
外部テンキー付属
外部101キーポート、外部FDDポート
ニッカドバッテリー(充電2時間、2時間使用
280W×220D×54H 4kg

80386/33MHz セット 超特価¥398,000

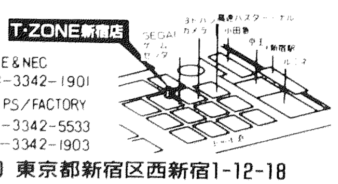
RAM 4MB
64KBキャッシュ
3.5"×1、5"×1
80MB HDD
スーパーVGA 1MB SVGAモニター
2シリアル、1パラレル
1ゲーム
マウス
101キー

Microsoft BALLPOINT MOUSE 超特価¥28,000

VGA信号をビデオに変換するコンバーター 超特価¥98,000
320×200～640×480の信号をS-VHS、ビデオに変換して、テレビやビデオに接続できます。

MINI-SCSI 超特価¥29,800
プリンタポートにつなぐだけでSCSIデバイスが使えるようになります。
87 MB HDDセット 超特価¥98,000

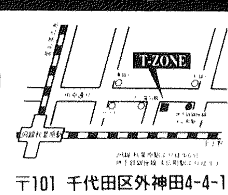
T・ZONE 新宿店 ☎03-3342-5533



1F APPLE & NEC 03-3342-1901
2F IBM PS/FACTORY 03-3342-5533
FAX 03-3342-1903
〒160 東京都新宿区西新宿1-12-18

T・ZONE 本店 ☎03-3257-2650

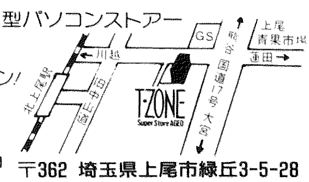
2F MICOMゾーン ☎03-3257-2649
4F PS PLAZA
3F IBMゾーン ☎03-3257-2652
FAX 03-3257-2661



〒101 千代田区外神田4-4-1

T・ZONE 上尾店 ☎048-774-0511

スーパー型パソコンストアー
上尾に オープン！
営業時間 AM10:30～PM7:00



〒362 埼玉県上尾市緑丘3-5-28

※T・ZONEの7Fは専門書籍（国内・外）・データーブックの専門フロアです。 大量在庫中！ ほしい本は何でも揃います。

〈資料請求番号003〉

Learning C

Cがわかる
うまくなる学習方法を探る

特集

Cの学び方・教え方

現在のパーソナルコンピュータのアプリケーション開発には，おもにC言語が使われている．C言語を使用することの是非はともかく，プログラマとして生活していくためにはC言語を身に付けることが必須といってもよい状況であることはまちがいない．しかし，C言語は学ぶのが難しいといわれる言語である．なぜC言語は難しいのだろうか？　C言語を学ぶコツはないのだろうか？　また，新人プログラマにC言語を教える場合，どのように教育を行えば，効率よく学習を進めさせることができるのだろうか？

今回の特集では，C言語の学び方から教え方まで，プログラミング学習のよりよい方法を模索してみた．

Teaching C

Learning C

コンピュータへのものの頼み方

わかりやすいこと、丁寧なこと、これがいちばん大事

株式会社ジャステックラボ
山本博昭

「Cの学び方・教え方」という特集の冒頭にページをいただけることになった．実際の話，この特集のタイトルをうかがった時,「書くよりも読みたい」と思ってしまった．ソフトウェアというものを書いて日々の食事にありつけるようになってから10年以上の月日が流れたが，いまだにC言語に限らずプログラムへの道は平坦なものではないというのが実感である．今回は，こんな筆者の経験の中から，プログラムを書く場合の進め方や検証の手順などを通じて，これからプログラミングを志そうとしている方やはじめたばかりの方へメッセージをお届けしたい．

Learning C

機転のきかない奥様とサトリのお化け

頭の中の要求をどう表現するか?

仕事から帰って，ひと風呂あびる．居間であぐらをかいて家人に目くばせすると，よく冷えたビールと気のきいた肴が運ばれてくる……というような理想的な環境で生活をしてみたいものであるが，現実はそう甘いものではない．「あうんの呼吸」といわれるコミュニケーションがあるが，実際には「おーい，お茶」というと紅茶が出てきたり，「あれ取ってくれ」というのと鉄アレイが出てきたりするのではないだろうか（これはあくまでも想像の世界であり特定のモデルは想定していない）．

親子や親友，恋人や夫婦といえど，最初から以心伝心というわけにはいかない．一流のバーテン氏などのように，初対面の客でもその人が何を欲しているのか見当がついてしまうという場合もあるが，通常の場合は「こういう状況の時に，この人がこういうサインを送ってくるのは，あの要求のことなのだ」という経験の積み重ねがあってこそのコミュニケーションなのだ．

以心伝心といえば，「サトリのお化け」といういい伝えがある．このお化けは人の心の中を読み取ってしまうそうで「いまおまえは，心の中を読み取られたくないと思っていることを隠そうとしているな」などという性格のよくなさそうなやつである．「サトリのお化け」に対抗する手段として有名なのは，まとまったことを考えないようにすることらしい．いくら人の心の中が読めても，その人が何1つまとまったことを考えていなければ何の意味もない．退屈した「サトリのお化け」は早々に退散するというわけだ．

さて，我われがいま相手にしようとしているのは，コンピュータである．この機械は以心伝心どころか「サトらないお化け」である．とにかく，いっさいの融通というものがきかない．このような機械を相手にプログラムという形で物事を頼むためには，次のことを肝に銘じて，絶対に短気をおこさないことが第1の条件である．

①コンピュータは，いわれたとおりにしかやらない．

プログラム開発の現場で，よく「あれぇ，何でこんな結果になるんだ？」などという声を聞く（筆者もよくいっていると思う）．ところが，コンピュータにとってみれば，たとえプログラマの意向にそぐわなかったとしても，いわれたことをそのまま実行しているだけなのだ．それがなぜ意向にそわないかというと，プログラマがまちがった指示を出したからにほかならない．プログラムは書いたとおりにしか動かない．バグを書けば，バグのとおりに動くのが当然なのである．コンピュータは，書いたプログラムのとおりにしか動かない……．

②コンピュータは，いわれたことしかやらない．

①と同じことをいっているようだが，少々ニュアンスが違う．たとえばテキストエディタを作ったとしよう．文字入力中にだんまりになることもなくなり，検索や置換もスムーズに動いている．安心してプログラム開発に使いはじめたが，ある日なにかの拍子に「強制終了コマンド」を入力してしまった．2時間かけて作ったソースが一瞬のうちに闇の中に消えていった……．エディタは作ったとおりに動いていた．ただ，強制終了コマンドに「ほんとうに終了していいのか？」という確認動作を入れ忘れたのである．コンピュータは，気をきかせたりはしない．コンピュータはいわれたことしかやらない……．

③コンピュータは，やめろといわれるまで繰り返す．

筆者の以前の同僚に，パソコンのリセットボタンに自筆で「終了キー」と書き込んでいた者がいるが，ご存知のとおりコンピュータは，終了判定のない繰り返しを書いて実行させると，いつまでも同じことを繰り返すことになる．疲れたとか無駄だからなどといって適当なところで打ち切ったりはしない．世にいう「永久ループ」である．これは素人目には「暴走」と混同されやすいが，厳として区別すべきものである．知ったかぶりをして「これ，暴走しているんですけど」などというと恥をかくことになるかもしれない．コンピュータはやめろといわれるまでやめない……．

上の3項目を見出しだけ見ていると，何だか子供に用事を頼むときや何か教えるときの注意書きのようである．いってみればコンピュータは，「とてつもなく根気があって従順な」子供なのである（実際にはそういう子供などいないことは歴史が証明しているが）．そう考えれば，自分の作ったプログラム（要求）がうまく動かないからといって，腹を立てたり途方にくれたりしてはいけない．とにかく筋道立ててわかりやす

く正確に物事を頼まないと，決してコンピュータは，自分の思いどおりの仕事をしてくれないものなのだ．

それでは，いかにしたら思いどおりのプログラムを書くことができるようになるか．コンピュータの電源を入れるのはもう少し待って，それを考えていくことにしよう．

Learning C
キーボードを叩くのはちょっと待って
論理を整理しないで書くと失敗する

プログラムをスムーズに作るコツは，自分がコンピュータに何をしてもらいたいのかを明確に把握することである．コンピュータに限らず，何かを相手に頼むときに自分自身がその要求をぼんやりとしか考えていなければ，十分な結果を得られる見込みはない．ましてコンピュータは先ほど書いたように「超」のつく融通きかずなのだ．「なんとなくこういうのを書いてみようかな」などとはじめたプログラムは必ず失敗すると心するべきである．

たとえば「大文字を入力したら，小文字にして表示する」というプログラムを書いてみたいと思ったとしよう．これは一見すると，かなり明確で具体的な計画のようにみえる．ところが，これを書いてやろうとはりきってキーボードに向かうと，たぶんうまくいかないだろう．なぜならば，この計画はコンピュータから見たら，ちっとも具体的でないからである．

コンピュータの声

大文字を入力して？　大文字って何なのさ．入力って何からどうやってやるの？　小文字といわれたってわからないし．表示はどういう体裁でやるんだろ．

計画を立てた者のいい訳

大文字・小文字っていったら，アルファベットのＡとａとかに決まっているだろうが．入力は普通キーボードからやるんだし．表示の体裁かぁ．そういえば細かいところは考えていなかったけど，１行入力したら，そのまま大文字のところを小文字にしただけで素直に表示してくれればいいや．

コンピュータのさらなる声

最初からちゃんとそういってくれないとわからないんだってば．でもまだ釈然としないところはあるな．大文字を小文字に変えるのはいいけど，ほかの文字はどうするんだ？　そのまま表示するのか，無視して表示しないのか．

もちろん，世間のコンピュータはこういうふうに指摘してくれたりはしない．だから，キーボードに向かって「void main(void)<RET>{<RET><TAB>」と威勢よくタイプしたところで「うー……」という結末になる確率が高い．「なんとなくこういうことがしたい」と思うことが悪いのではなく，それをそのままの状態でプログラミングに移してしまうことが失敗の原因となるのだ．

では，さきほどの「大文字を入力したら，小文字にして表示する」というアイデアをプログラムにするときの進め方をもう一度考えなおしてみよう．ただし，今度はコンピュータの影の応援はなしである．

①やってみたいことを言葉で口に出して説明できるくらいに明確にする．頭の中でぼんやりと「こうしたいな」と考えているうちはプログラムにならない．

[解説]
これに関しては，「大文字を入力したら，小文字にして表示する」という具体的な要求があるのでいいだろう．問題はこのあとなのである．

[ポイント]
ノートを用意して，新しいページのいちばん上に明確になった計画を書いておく．

②プログラムはコンピュータ上で動くものであるから，これから書こうとしているものが実際にどうコンピュータで動作するのかイメージしてみる．

[解説]
コンピュータ上で，「大文字を入力したら，小文字にして表示する」プログラムを使うときに，どのような操作をして，どのような結果を出すのかを考えてみる．あくまでも具体的に，そのプログラムが動いている様子を想像するのがコツ．実例をあげてみると……

- プログラムを起動すると，「英文を１行だけ入力してください」と表示する．
- キーボードから１行入力する．
- その下の行に，大文字のところを小文字にしたものを表示する．

[ポイント]
さきほどのノートに，できるだけ具体的に箇条書きにする．取り扱い説明書を書くような気分で．

③実際に動いているイメージが固まってきたら，今度はプログラムを作る立場から気になることを書き込んでいく．

Ｃプログラミング用語
ASCII　American Standard Code for Information Interchange（米国標準情報交換用符号）の略．７ビットで表現できる文字セットのこと．

[解説]
ここではじめてキーボードに向かっている自分の姿を想像する．ただし，まだ想像だけである．さきほどの具体的なプログラム動作のメモに，今度はプログラムを書く立場になって注釈を入れていく．この段階で，プログラム的に無理がある(現段階のプログラミング技術ではできないとかシステム上の制約でとか)ことがわかるかもしれない．その時は，②に戻ってプログラム動作から考えなおす．とにかく，実際にプログラミングする時のイメージをできるだけ現実的に固めていく．キーボードに向かいたい気もちをグッと押さえて，ここはじっくり取り組んでいただきたい．では，さきほどのノートに「＊」マークを付けて注釈を書き込んでみる．

・プログラムを起動すると,「英文を１行だけ入力してください」と表示する．
　＊画面のイメージは

```
A>ｺﾏﾝﾄﾞ
英文を１行だけ入力してください
■
```

・キーボードから１行入力する．
　＊リターンを押した時点で入力が終了するものとする．

・その下の行に，大文字のところを小文字にしたものを表示する．
　＊１行入力した文字列を１文字ずつ判断して'A'〜'Z'なら'a'〜'z'として表示する．
　＊'A'〜'Z'とは，ASCIIコードで41hから5Ahの範囲である．
　＊小文字にするためには，ASCIIコードで'a'(61h)と'A'(41h)の差を補正してやればよい．
　＊'A'〜'Z'以外の文字は，そのまま出力とする．
　＊画面のイメージは

図１　例出のプログラムの計画

【着想】
「大文字を入力したら、小文字にして表示する」

【プログラムの具体的な動き】

(1) プログラムを起動すると、「英文を1行だけ入力してください」と表示する.

(2) キーボードから1行入力する.
　リターンを押した時点で入力が終了するものとする.

(3) その下の行に、大文字のところを小文字にしたものを表示する.

　1行入力した文字列を1文字ずつ判断して'A'〜'Z'なら'a'〜'z'として表示する.
　'A'〜'Z'とは、ASCIIコードで41hから5Ahの範囲である.
　小文字にするためには、ASCIIコードで'a'（61h）と'A'（41h）の差を補正してやればよい.
　'A'〜'Z'以外の文字は、そのまま出力する.

【出力画面イメージ】

```
A>ｺﾏﾝﾄﾞ
英文を1行だけ入力してください
aBCDeFG!!
abcdefg!!
A>
```

Cプログラミング用語
ANK　Alphabet（英字）・Numeric（数字）・Kana（カタカナ）を意味し，８ビットで表現できる文字セットのこと．

```
A>コマンド
英文を1行だけ入力してください
aBCDefG!!
abcdefg!!
A>
```

[ポイント]

プログラムの動きを箇条書きにしたら，それをもう一度よく読んで，思いつく限りの注意事項を書き込んでいく．プログラミングに慣れてきて大規模なプログラムを書くようになると，関数ごとの設計やモジュールの分割といった計画もこのへんで固まってくるようになる．

いままで書いてきたノートをまとめたのが図1である．これが「プログラム仕様書」となる．ここまで具体的に書いておくと，次のステップのプログラミングがスムーズにいくだろう．

Learning C
Cだからむずかしいのかどうなのか
知らなくていいことと悪いこと

自分のプログラムについて整理もついた，処理の手順もすっかり見通しがついた．そうなればいよいよキーボードに向かってプログラムを作る段階になる．この時点までには当然なにがしかのCコンパイラが手元にあるはずである．近ごろのCコンパイラは，「統合開発環境」といって，ソースを作成してコンパイルをし，エラーを修正してデバッグまでという一連の作業が，1つの環境の中でマウスを使ったりして快適にできるというのが常識になっている（バグを取ってくれるようになればいうことはないのだが……）．

そんな処理系の重装備に合わせて，マニュアルの方も増大していく．読者の中にも，買ってきたCコンパイラの箱を開けて，そのマニュアルの山にため息をつかれた方もいるに違いない．筆者の手元にも，あるCコンパイラのマニュアルがうず高く積まれている．

さて，これからCをはじめよう，あるいははじめたばかりの人で，このマニュアルの山をすべて読破してからプログラミングを……などと考えている方がいたら，ぜひやめていただきたい．ベテランになってコンパイラや統合環境の機能を自在に使いこなすようになった人ならいざしらず，最初からこのマニュアルをすべて理解しようというのは，いかにも無理がある．とくに，ずらりと関数が並んだ「関数リファレンスマニュアル」などを見て，全部覚えなきゃならないとは絶対に思わないことだ．そんな暇があったら，まずプログラムを書くにあたってのミニマムなルールを覚えて，とりあえず自作のソフトを動かしてみることである．自分で書いたプログラムがコンピュータ上で動作することの楽しさを知ったら，あとは使える関数を1つずつ増やしていけばいい．要は，いきなりCの機能をすべて覚えようとしないことだ．C言語にはプログラム記述を支援するたくさんの機能があるが，これを知らないとプログラムが書けないという項目はそれほど多くない．最初は知らなくてもいいことを全部覚えようとするあまり，Cをあきらめるというのは本末転倒というものである．

ここでは，前節で整理のついたプログラム計画を実際のプログラムに移しながら，プログラムを作るにあたって「これだけは」というものがどのくらいあるかを見ていくことにしよう．

論理の表現方法を2つ覚える

まず，プログラムを書くにあたって必要な論理表現（構文）をおぼえる．Cの構文はたくさんあるが，とりあえずは「分岐」「くり返し」を使えるようになれば，プログラムは書けるようになる．書けるようになってから，より見やすく・より美しく論理を表現できる構文を仕入れることにしても遅くはない．

・条件によって処理を変える．

（……ならば……する，そうでないときは……する）

```
if(条件)
  {
    その条件に適合したときの処理
  }
else
  {
     その条件に適合しなかったと
     きの処理
  }
```

・条件によってくり返す．

（……であるかぎり……する）

```
while(条件)
  {
      条件が成立するかぎりくり返
      す処理
      途中で打ち切りたい場合のた
      めに「break;」という予約語が
      ある．
  }
```


Cプログラミング用語
BCD　Binary Coded Decimal（2進化10進）の略．桁数が大きくなると本来の10進数とは多少の誤差が生じる．

ライブラリ関数は，マニュアルの索引に頼り切る

Cのライブラリ関数は，ため息の出るほどたくさんある．全部覚えようと思ったらたいへんな労力である．しかし逆に考えると，これだけの数があるのだから，プログラムを組むときに心強い味方になってくれる．「こういう関数が欲しいな」と思ったら，ちゃんとあるというのはうれしいものだ．

マニュアルの山を全部読む必要はないといったばかりだが，ライブラリ関数のリファレンスマニュアルはおぼえるものではなく，「使いこなす」ものだと思って欲しい．まず索引を開いてみよう．キーボード入力関係の関数にはどんなものがあるか，ファイル操作の関数は，という具合に必要に応じて使い込んでいく．何度もいうようだが，暗記する必要はない．それどころか，うろ覚えで関数を使ってしまうのはたいへん危険なことである．引数を勘違いしたままプログラミングしてしまうと，当人が思い込んでいるだけにまちがいの発見がたいへんになる．ある関数を使うときに，さっとマニュアルがひけて，それが苦痛にならないくらいに索引に精通することが大切である．

C　O　L　U　M　N

参考　ソースの書式について

Cのような不定型の書式を許している言語は，必ず「どう書くか」という論議が出てくる．とくに字下げ（インデント）の問題は，好み・主義・理論が入り乱れての大論争になる傾向にある．本文にあげたif文にしても

```
if(a > 0)  {
    plus_proc();
}
```

こう書く人がいたり，

```
if(a > 0)
    {
    plus_proc();
    }
```

こんなふうに書く人もいる．ここでは，あえてどれがいいという主張はしないでおこう．社内的な規定もあるだろうし，個人的な趣味の場合もある．ただ，その場その場でいろいろな書き方をするのだけは避けたほうが賢明である．ひどい場合には，1本のソースファイルの中にさまざまな書式が入り乱れてしまったりしているものもあるが，見づらいだけではなく，思わぬトラブルの原因になりかねない．

サンプルプログラムなどを最大限に活用する

何か物事をはじめるときの常として，「人の真似から」というのがある．プログラムの書き方につまったり，関数の使い方がどうしてもわからないときは，マニュアルのサンプルプログラムや参考書を最大限に活用することである．身近なところにCの達人がいればいうことはない．

それでは，さきほどの「大文字を入力したら，小文字にして表示する」というプログラム計画をじっさいにプログラミングしてみよう．

①プログラムを起動すると,「英文を1行だけ入力してください」と表示する.

文字列を表示する関数はさまざまあるが，ここでは「printf」を使ってみる．

②キーボードから1行入力する．リターンを押した時点で入力が終了するものとする．

キーボードから1行入力する関数「gets」を使う．この関数は入力のためのバッファを用意する必要があるので，これを変数として定義する．バッファをどのくらい取るかも考慮しなければならない．

③その下の行に，大文字のところを小文字にしたものを表示する．

gets関数によって入力した1行分の文字列を1文字ずつ切りだして，それが'A'～'Z'かどうか判定する．該当していたらコードの差を補正するための計算をしてから表示する．文字列の終端である'¥0'が出てくるまでのくり返し文の中にif文が入るスタイルとなる．

Cプログラミング用語
BIOS　Basic Input and Output System（基本入出力システム）の略で，システムコールで呼び出すことができる．

図2はプログラミングした実際例である．前記の文章と実際のコーディングを対照して，計画の流れと実際のプログラムが一致しているかどうか検証していただきたい．完成したプログラムリストは図3に掲載する．

Learning C
楽に書く・うまく見えるC考
Cの魅力はこんなところにも

試しに作ってみたプログラムがなんとか動くようになった．問題はここからである．「動いた，動いた」と喜んで酒を呑みに行き，きれいさっぱりと終わったことにしてしまっては進歩がない．動いて(しまった)プログラムでも，もっといい書き方はないかという姿勢をつねに持つのが上達の秘訣である．

とはいうものの，求道者のように修業ばかりしていてはおもしろくないし，つらいばかりではうまくならないという筆者の主張（単に楽天的なだけだという説もあるが）にそって，ここでは「なるべく楽にプログラムを作るための心がけ」「一見うまく見えるCの機能」「Cを知らない人に尊敬されるポイント」などを紹介してみよう．

関数の分割・モジュールの分割

さきほど例として作ってみたプログラム程度のものなら，いきなりmain()……などとはじめてしまっても一気に書き切ってしまう分量だが，次第に複雑で大きなプログラムを書いていくと，関数やモジュールの構成を考えて

図2　「大文字小文字変換」プログラムのコーディング解説

```
: #define        BUFSIZE        160
```
キー入力のためのバッファのサイズ．「1行だけ」と書いてあるので，余裕を見て2行分もあればいいだろうと思った．このように，疑似命令を使って定義しておくと，修正もしやすくなる．

```
: void main(void)
: {
```
今回は，main関数のみ．

```
:     char    StringBuffer[BUFSIZE];
```
キー入力のためのバッファ．サイズは先に定義してある．

```
:     char    *sptr;
```
上記のバッファのポインタ．

```
:     char    c ;
```
判定用に1文字入れるための変数．

```
:     printf("英文を1行だけ入力してください¥n");
```
文字列表示を行い，改行する．

```
:     sptr = gets(StringBuffer) ;
```
キーボードから1行入力する．入力された文字列はバッファに格納され，その先頭文字をさすポインタが返ってくる．画面は1行改行される．

```
:     while((c = *sptr++) != '¥0')
:     {
```
入力された文字が終端になるまで以下の処理を続ける．

```
:         if((c >= 'A') && (c <= 'Z'))
```
その文字が'A'～'Z'の範囲にあるかどうかの判定．

```
:             c += 'a' - 'A' ;
```
条件に合致していれば，小文字に変換するための計算を行う．
（そうでなければ，何もしない）

```
:         putchar(c) ;
```
1文字画面に表示する．

```
:     }
```
くり返し処理の終端

```
: }
```
プログラムの終り．

Cプログラミング用語
インクルード　#include前処理指令などでヘッダやソースファイルをプログラムに挿入すること．

いくことが大切になってくる．

以前，ある人からCのプログラムを見せてもらったとき，数百行のプログラムが1つの関数，つまりmainだけでできているのでめまいを感じたことがある．ソースを画面で見ても，延々と続くのはmain関数ばかりで，前のほうで何の処理をやっていたのか忘れてしまいそうである．たしかに，C言語では1つの関数が何行以内などという規定はないし，動いているのだからこれでいいじゃないといわれたらそれまでだが，もし何かの都合でこのプログラムに手を入れなくてはならなくなったら……筆者は遠慮したい．

こういうプログラムができ上がってしまう大きな原因は，最初から何度もいっているように，プログラムの構想を細部まできちんと具体化しないでいきなりキーボードに向かい，思いつくままソースを書いてしまうことだろう．書いているうちに，「あ，この処理もいるな」「あ，ここでこれをやっておこう」などとやっているので，全体の構成などおかまいなしに関数がどんどん太っていく．しかも，これではおそらくまともに動かないので，動かしてみてあちらこちらに修正を入れて……という結果である．

ノートにプログラムの動きを具体的に書く作業に慣れてくると，個々の動きをパーツ化して考えられるようになってくる．「この部分の処理はこういう関数を作る」「この部分は共通に処理できる」といった発想が自然に出てくるのだ．最初のうちは，1つの関数の長さをなるべくエディタで見て1画面におさまるように書くべきである．どうしても無理というなら，せめてプリンタに出力して1ページ．それでもおさまらない場合は，関数の仕様に問題があると思ってよい．

プログラムのモジュール構成に気を使う最大のメリットは，「貯金」ができることである．何本もプログラムを書いていけば，必ず似たような処理が必要になる場面が出てくる．そのたびに同じような処理関数を書くよりも，過去に自分が書いたものを使えるようにしておけば，能率はぐんとアップする．しかも実際に動いていた関数を使うのだから，安全確実である．このような「関数の貯金」を作るためには，いま書いている関数を貯金するという明確な意識をもつべきである．本物のお金もそうだが，もって歩いている財布の中で何となく使い残した金額がたまっていくことはめったにないことだ．関数の貯金を作るためには，以下のことに注意するとよい．

- 特有のプログラムのみに必要な処理はなるべく固めておく．
- 呼び出し側で操作できるような処理を考える．
  (エラーメッセージは関数内にもたないで，呼び出し側で指定するようにするなど)
- できるだけシンプルな機能にしぼる．
  (複数の機能をもたせるよりも，それを分解しておいて，呼び出し側で組み合わせて使えるようにしたほうがよい)

図3　サンプルプログラム「大文字小文字変換」

```
#include <stdio.h>
#define    BUFSIZE     160

void main(void)
{
    char    StringBuffer[BUFSIZE];
    char    *sptr;
    char    c ;

    printf("英文を1行だけ入力してください¥n");
    sptr = gets(StringBuffer) ;
    while((c = *sptr++) != '¥0')
    {
        if((c >= 'A') && (c <= 'Z'))
            c += 'a' - 'A' ;
        putchar(c) ;
    }
}
```

このように考えていくと，さきほど書いたようなmainだけが何百行も続くようなプログラムは，書き捨て使い捨てになる公算が強い．関数の一部をぬき出して，次への資産として生かすのは非常に困難なことである．

Cな書き方

Cのことを「高級アセンブラ」などと呼んでいた人がいたが，たしかにCにはユニークな機能がある．この手の

Cプログラミング用語
インデント　行の先頭にタブや空白を入れること．プログラミングスタイルのひとつでソースファイルでプログラムの流れを判断しやすくする．

機能を使ってみると，気分的にCをやっているということが実感でき，あまりCを知らない人に見せても感心してもらえる（かもしれない）．

■インクリメント／デクリメント演算子（++と--）

Cが高級アセンブラと呼ばれた大きな原因の1つに，この演算子のようにCPUの動作が連想できるような機能があることがあげられる．インクリメント／デクリメントというのは，ご存じだとは思うが，「1つ増加」「1つ減少」のことである．通常なら「a = a + 1」と書いてもいいところを，わざわざ特別な演算子まで作って「a++;」などと書かせるのは，CPUにインクリメント／デクリメントの専用命令があるからに違いない．

ところで，この演算子は下のような使い方に限っていえば

```
abc++;
++abc;
```

どちらでも結果は同じである．ところが，図4のプログラムを動かしてみた場合に違いが出てくる．「i++ < 5」と書くと比較した後でiを+1し，「++i < 5」と書くとiを+1した後で比較をするためである．こういったところにもこの言語の「Cなところ」が出ている．

■三項演算子

まず，下の式を見ていただきたい．「三項演算子」が何者なのか知らない人にとってはほとんど暗号の世界だと思う．

```
abc = (def >= 0) ? 10 : 20 ;
```

この式を文章で説明すると以下のようになる．

変数defの内容が0以上であれば10を，0未満であれば20を，変数abcに入れる．

つまり，上記の式は次のプログラムと同じことなのである．

```
if(def >= 0)
    abc = 10 ;
else
    abc = 20 ;
```

こういう表現もまさに「Cな書き方」であろう．C言語の特徴に，なるべくシンプルに（ありていにいってしまえば，ソースを入力するときに楽なように）記述できるという傾向があることを指摘しておこう．Pascalなら BEGIN～ENDとていねいに書くところも，Cは「{}」などとすませてしまうし，Cのプログラマは，「a = a + 10」とは書かない．「a += 10」なのである．こうした特徴ある書き方にも「Cはむずかしい，とっつきにくい」といわれる一因があるのは事実なのだろうが，これを「Cな書き方」として楽しんでこそCの達人をめざすプログラマの姿勢といえるであろう．

■再帰呼び出し

これはC言語特有の機能ということではないが，プログラムを書いていく上でぜひ使いこなしていただきたいテクニックの1つである．再帰呼び出しの原理は簡単である．「自分自身を呼び出す」，つまり

```
tango()
 {
   tango();
 }
```

こういうことである．このプログラムだと，はてしなく自分を呼びつづけて終わることのないプログラムになってしまうが，何かの処理をしていて，

図4　インクリメント演算子の動作とその実行画面

```
void main(void)
{
    int i;

    i = 0 ;
    while(i++ < 5)
        printf("%d¥n",i) ;
    i = 0 ;
    printf("¥n");
    while(++i < 5)
        printf("%d¥n",i) ;
}
```

●上記プログラムの実行画面

```
1
2
3
4
5

1
2
3
4
```


Cプログラミング用語
後判定反復文　制御文の一種で，do～文のように継続条件を後ろで判断する文．

図5　再帰呼び出しの実例(ユークリッドの互除法)

```
int euclid(int x , int y)
{
    if(y == 0)
        return x ;
    else
        return euclid(y, x % y);
}

※再帰呼び出しを使わないで書くと……

int euclid(int x , int y)
{
    int temp ;

    while(y != 0)
    {
        temp = x % y ;
        x = y ;
        y = temp ;
    }
    return x ;
}
```

その途中結果をもとに最初から同じ処理をする場合などにはこの再帰呼び出しがおおいに威力を発揮する．図5は，最大公約数を求める「ユークリッドの互除法」をCでコーディングした例である．再帰呼び出しを使えば，こんなにすっきりとコーディングできる．

プログラムも1つの「書きもの」である．同じ書くなら美しく読みやすいものを書きたい．そして他人が見てもそう思えるものになってほしい．そのためには，「文体」というものを考える必要がある．軽いエッセイにはそれなりの，技術論文にもそれなりの文体があるように，Cで書くプログラムにもCならではの表現方法があろう．ただし，エッセイも論文もそうであるように，Cの表現方法にも「唯一無二」というものはないので，これは経験を積むことによって次第に身につけていくことになると思う．

Learning C

やっぱり動かなかったというときは

パニックにならない検証法

さて，プログラムの計画・作成とスムーズに進めそうな気がしてきた．ところが，運命というものはあまり親切なものではなく，どんなに工夫して仕様書を書き，どんなに注意してソースを書いても，やはり「あら？」ということになってしまうのが世の常である．ソフトウェア技術者と呼ばれるプロもみんな「あら？」となる．私もこの業界の末席を汚しているが，ときどき「ご職業は？」の問いに「デバッグです」と答えそうになる．

プログラムが書いたとおりに動かない場合，おっと失礼，書いたとおりには動いているものの，書いた者の最初の意図と違っている場合，「デバッグ」という作業が必要になる．デバッグといえば，「デバッガ」あるいは「エミュレータ」などというマシンデバッグ手段を連想しがちだが，その前に「机上デバッグ」という言葉を再認識してみたい．

パソコンでのプログラミングのように，ひとりが1台のマシンを専有でき，好きなときに好きなだけ使えるという状況では，あまりピンとこないかもしれないが，多人数でマシンを共有する場合や，賃貸で1分いくらなどとCPUが非常に高価な場合などは机上デバッグやデバッグのプランニングは非常に大切である．また専用に使うことのできるマシンの場合でも，ただやみくもにデバッガを起動してマシンの前に貼りついているよりも，計画をたててデバッグを進めていったほうが能率があがるものである．ここでは，せっかく書いたプログラムが予定どおり動かなかったときの検証方法を考えてみよう．

机上シミュレーション

動かないプログラムが小さいものならば，一度はこの「机上シミュレーション」を体験していただきたい．用意するものはノートとペンとプログラムリストである．机上シミュレーションは，自分がコンピュータに変身したつもりになって，プログラムを実行していく．

①ノートにプログラムで使用している変数名を書いていく．あとから値を書き込んでいくので，その分の余白をあらかじめ空けておくとよい．

②main関数から実行を開始する．パラメータが必要な場合は，適当な値を想定する（極端な値を想定して，

Cプログラミング用語

エスケープシーケンス　ソースプログラム作成時，文字列制御文字を記述するために使われる．0x1b(ESC)や0x5c('¥')に続く表記方法

パラメータエラーとなる過程をシミュレーションすることもある)．変数に動きがあった場合は，逐次その値を該当の変数のところに書き込んでいく．

机上シミュレーションのメリットは，いやがおうでもじっくりとリストを見ることになることである．通常のリストチェックでは，どうしても読み飛ばしてしまいがちな単純な書きまちがい（自分は正しく書いたつもりという先入観があるので，こういうのがいちばん発見しにくいものである）などが見えてくる．リストを目で見ながら頭の中で変数の変化を追っていけるようになるまでは，自分自身を「超低速マシン」と化して確実に検証していくのが早道といえよう．机上シミュレーションを一度行うと，いま使っている少々旧型のパソコンの悪口が自然と減るという思わぬ余禄も出てくる．自分がコンピュータのかわりをすると，いかにこの機械が高速であるかわかるというものである．

デバッグのプランニングと勘どころ

さきほども書いたように，書いたプログラムが予定どおり動かないからといって，すぐにデバッガを起動して，やみくもにデバッグをしようとしても，なかなか能率はあがらない．デバッグの巧拙は，デバッガの使い方に通じているかどうかということではない．プログラムの動きがおかしい……そう感じた時からデバッグははじまっているのだ．

デバッグのサイクル

図6のように書いてしまうと終りがなくなってしまうが，何周も回って一段落というのが現実であろう．このサイクルの中で，プログラムを書きはじめて日の浅い方にはあまりピンとこないかもしれない2つの段階を少々説明したい．

「不具合の観察」

書いたプログラムが動かない，おかしいとなった時，まず何をしようとするだろうか．いきなりプログラムリストを開いたりするのはよくない反応である．だからといって，見なかったことにする，踊りながら部屋を出ていくなどというのもいけない．現実を冷静に受け止め，「おかしくなり方」をよく観察する．その後のデバッグの手間は，この観察によって大きく変わってくることを十分に心してほしい．

- 必ず起きるのか，ある特定の状況で起きるのか？
- 特定のデータの時におかしくなるのか？　どんなデータでもだめなのか？
- 動作環境によって症状に違いがあるか？
- ディスクドライブなどによる症状の違いはあるか？
- （もし使える状況なら）他の機種でも発生するか？
- 確実に再現する操作は何か？

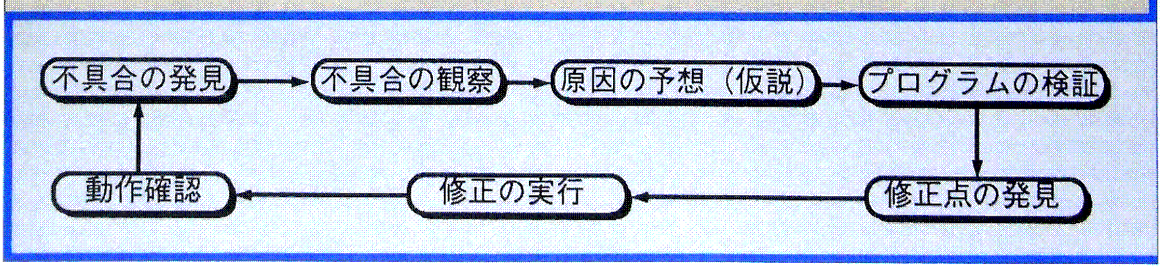
「こういうデータの時に，こういう操作を行ったら，こうなる」という確実に再現できる現象をつきとめることができれば，ゴールは近い(かもしれない)．現象が特定できれば，次に書く「原因の予想」もつきやすくなる．

「原因の予想」

不具合の現実をよく把握したら，今度は「なぜそうなるのか？」を予想だててみる．この時に大切なのは，自分のプログラムをよく理解するということである．何をあたりまえのことを，と思われるかもしれないが，自分が書いたプログラムの動きを完全に理解していないことには，これから先の作業がはかどらない．逆にいえば，デバッグがはかどらないときは，プログラムの理解がよくできていなかったとして反省の余地があるのかもしれない．

確実に再現できる操作が確認できた

図6　デバッグのサイクル



Cプログラミング用語
エミュレーション　あるシステム上で別のシステムを模倣すること．数値演算プロセッサを持たないシステムでは数値演算エミュレータを使う．

ら，その操作をプログラムの処理の流れにあてはめていく．どの操作の時にどの関数の処理が動いているのかという観点から，不具合の原因となっている関数を特定していく．注意しなければならないのは，実際の症状を発生させている関数が必ずしも真犯人だとは限らないことである．前のほうの関数でおかしなパラメータが渡ってきていたり，どこか領域を壊していたりするかもしれない．ここで重要なのは，この関数が悪いと決めつけることではなく，そこにいたる過程で正しくはこうなっていなければならないのに，なぜならないかの仮説をたてることである．そして，この仮説をたてる過程で，要注意の箇所がリストアップされてくるはずであり，その箇所が本来はこう動かなくてはならないというチェックリストになる．さぁ，ここまでやったら晴れてデバッガの起動である．幸運を祈る……．

ちょっとお節介

ある不具合を追っているうちに，別のバグを発見してしまうことがよくある．そのとき，おもわずそのバグを追ってしまうのが人情というものだが，プログラムを書きはじめた段階ではこれは決してやらないほうがいい．せっかくデバッグ計画をたてて，ある程度の見通しをつけてはじめたのに，そこから脇道へ入ってしまうと何も準備していないのと同じことになってしまうからである．追いかけたい気もちをグッと押さえて，何事もなかったかのように（記録はちゃんと取っておくこと），計画どおりにすすめてほしい．能率もさることながら，プログラム開発にはこういった冷静さがなによりも必要なのだから．

Learning C

プログラマと読書

プログラミングを上達させてくれる本はあるか？

Cをはじめるにあたって，参考になる本を取り上げてみようと思い，自分の書棚や近くの書店を見て回った．手もちのいわゆる「入門書」も何冊かあるし，書店に行けばすさまじい数の「C入門……」がある．これらをいちいち紹介していたら本誌の1か月分のページをすべて割いても足りないだろうし，どれが適しているかというのは，読む人それぞれに異なるものであるから，少し違った角度からプログラムをはじめようとする人への提案をしてみたい．

『古典』を読む

Cに限らず，プログラミングをはじめよう，はじめたばかりという方には，ぜひご一読していただきたい本がある．筆者が新人のころにはすでにソフトウェアエンジニアの間で「書法」，「作法」と呼ばれて親しまれてきた名著である．どちらも10年以上前の著作だが，プログラムの構造やスタイルについての考え方は普遍的なものである．

> 「プログラム書法」第2版
> Brian　W.Kernighan & P.J. Plauger
> （木村泉訳）
> 共立出版

プログラムの構造や表現，ソースへのコメントなどについての基本的なスタンスを身につけるための必見の書．各節の見出しには「なるほど」と納得したり，思わずニヤリとさせられたりする．

> 「ソフトウェア作法」
> Brian　W.Kernighan & P.J. Plauger
> （木村泉訳）
> 共立出版

こちらは，さらに実際的にフィルタや文字列操作などをはじめとするプログラミングテクニックとその思想背景の宝庫である．サンプルプログラムは「Ratfor」という FORTRAN ライクな言語で記述されているが，Cプログラマならすぐに役立つだろう．

そして，「プログラミング言語C」．かの有名なK&R本である．C言語はその構造の柔軟さゆえ機能拡張の連続であるが，この言語の原典ともいうべきこの本は一度は手にとっていただきたい．Cのエッセンスがこの本の中に凝縮されているといっても過言ではないはずだ．

> 「プログラミング言語C」第2版
> Brian W.Kernighan & Dennis M. Ritchie
> （石田晴久訳）
> 共立出版

入門書の選び方

書店のパソコンコーナーには，パソコン本体の解説書や各種ソフトウェア／OSの解説書などに劣らない数の

Cプログラミング用語
オーバーレイ　一度にメモリに入らない大きなプログラムを，いくつかに分割してメモリに読み込み，実行する方式

「C言語入門書」あるいは「C言語解説書」が並んでいる．はじめてこの種の本を買いに行った方は迷ってしまうことと思う．まさか全部買う人はいないと思うが，店頭でぱらぱら内容を見たうえで購入して，いざ自宅で開いてみると「やはりわかりにくい」「知りたいと思っていたことが書いていない」ということで肩を落とした経験がおありではなかろうか．

入門書や解説書を必要とする人のレベルや傾向が千差万別である以上，「一般的にこの本が絶対いい」ということはいえないが，書店で選択するときの手法を1つご紹介しておこう．プログラミングをした上で自分が苦労したことやわかりにくかったこと，あるいは使い方が複雑な関数などを2，3リストアップしてから書店に行くのである．あとは，同種の本が並んでいる中で，その項目や関数を解説しているところをチェックし，いちばんわかりやすい説明をしている本を選択するのだ．

アセンブラを覚えよう

Cをマスターする上で，最良なのは「Cをはじめる前に，まずアセンブラをマスターする」ということである．何やら冗談めいているが，これはほんとうである．逆にいえば，アセンブラを知らずしてCを完全に使いこなすことは非常にむずかしいといえよう．Cを書きはじめた多くの人が一度は戸惑うポインタを使っての処理も，実際にアセンブラレベルでどう動作するのかを考えれば理解しやすくなるし，メモリモデルの違いによるデータや関数のアクセス関係などもCPUの動作に関する知識があれば早く理解できることだろう．

筆者の知り合いにも（筆者自身のことではない），ちょっと複雑なCソースを見ると，アセンブラのコードを見にいってしまう者や，「めんどうだからアセンブラで書きたい」という者がいる（くどいようだが筆者ではない）．いますぐにとはいわないが，Cでプログラムを書きつづけるのであれば，ぜひともアセンブラレベルの知識を身につけていくことをおすすめする．

Learning C　おわりに

長々とお付き合いいただいたが，筆者がいただいたページも残り少なくなってきた．ここまで読んでいただいた方には心からお礼申し上げる．自分の経験の中から，プログラムを書く上での準備やプログラミング作業の進め方，デバッグの心得などを書いてきたが，Cに限らず，プログラムとは「コンピュータにものを頼むときのいい方」というのが実感である．生身の人間もそうだが，同じ事を頼むのでも口のきき方ひとつでまちがったものが届いたり，すねられたりするものだ．スムーズに何かを依頼するには，まず頼みたいことをわかりやすくまとめる，相手にわかるようにていねいに伝える，誤解が生じたら冷静に原因をつきとめてしかるべき処置をとる．ちょっと前のヒットソングではないが，「これがいちばん大事」なのである．

プログラムをはじめる人にとって，ライブラリ関数やOS，ハードウェアの知識と，これから覚えなくてはならないことがたくさんあるだろう．まして，今後もプログラミング言語の中心的存在に君臨し続けるであろうC言語なら，GUI環境やオブジェクト志向，マルチタスク環境などの新しい波が怒濤のように押し寄せてくる．そんな中にあっても，プログラムというのは，人間が作ってコンピュータが実行するという図式が変わらずある限りは，仕様の柱を明確に整理して，自分の考え方を筋道だてていくという姿勢は忘れずにもちつづけていただきたい．「自分はCでコンピュータを使う」のであって，「Cとコンピュータが自分を使う」ようにならないためにも．

最後のおすすめ

今回の特集の各所に，「ノート（紙）に書く」という場面があったが，これは実際に実行していただきたい．たとえば，1本のプログラムを書くために1冊というように決めて，最初の仕様案から具体的な計画になっていく過程，プログラミングの時に気のついたことや苦労したこと，デバッグの時の記録やコンパイルを待つ間の落書きなど，プログラムを作るときにいつも手元において，何でもいいから書き込んでいく．プログラムができ上がったときに，ソースリストには結果が残るが，後々になってくるとこのノートに記されている「経過」が財産になってくる．なぜ，この時この関数ではなくてこちらを選んだのか？　このロジックの他にこういうロジックも検討していたのか，などという情報は記憶の中で薄れさせていくのはもったいない．何年かしてノートもたくさんたまってくると，それはどんな名著よりも自分自身への大切なメッセージとなることだろう．

Cプログラミング用語
オーバーヘッド　演算などが実行されない時間のこと．C言語では関数呼び出しは変数参照よりオーバーヘッドが大きい．

特集 Cの学び方・教え方

Learning C

61のライブラリ関数と使い方のヒント

Cの基本関数をマスターする

虎岩　登

C言語でのプログラミングのポイントのひとつは，標準のライブラリ関数をいかに使いこなすかにある．だが，400を越えるライブラリ関数を頭からすべて覚えるのは現実的ではない．Cにはどんな機能をもった関数があるのか，そして，Cを効率よく学ぶには，どれから覚えればよいのだろうか．ここでは，数あるライブラリ関数の中から，よく使われる61の関数を基本関数として取り上げ，その使い方のヒントを紹介しよう．

Cのライブラリ関数は，マニュアルを見ると非常に多くの種類があります．機種依存のグラフィック操作用関数などを除いて標準的なものだけでも，数えようによっては400を越えるかもしれません．でも，実際のところ，普段使うものは数十にしか過ぎないと思います．その基本的な関数の使い方をきちんと覚えれば，かなりのプログラムを作ることが可能です．本稿では，そんな基本的関数を紹介していきましょう．

今回，紹介する関数を表1に示します．基本関数はこの表のようなグループ分けができると思います．それぞれのグループごとに関数の内容を見ていくことにします．

表1　Cの基本関数

入出力関数	fopen()，fclose() fgets()，fputs() gets()，puts() getc()，getchar() putc()，putchar() fseek()，ftell() fread()，fwrite()
書式付き入出力関数	printf()，scanf() fprintf()，fscanf() sprintf()，sscanf()
文字分類関数（マクロ）	isalpha()，isupper()，islower() isdigit()，isxdigit()，isalnum() isspace()，ispunct()，isprint() isgraph()，iscntrl()，isascii()
文字の変換関数	toupper()，tolower()
文字列操作関数	strcat()，strncat() strcmp()，strncmp()，stricmp() strcpy()，strncpy() strlen()，strchr()，strpbrk()
文字列－数値変換関数	atoi()，atol()，atof()
シフトJIS関係	jistojms()，jmstojis() btom()，mtob() iskanji()，iskanji2()
その他　強制終了	exit()，atexit()
その他　メモリ割り当て	malloc()，free()
その他　表操作	qsort()，bsearch()，lsearch()
その他　環境変数取得	getenv()

Learning C 入出力関数

入出力関数の分類

Cの入出力関数にはかなりの数がありますが，それをさらにいくつかの方法で分類することが可能です．ひとつの分類法は，「低レベル入出力関数」と「高レベル入出力関数」です．低レベル入出力関数は，ファイルハンドルという整数を使ってファイルにアクセスするのが特徴で，OSに近いところでの操作を行います．高レベル入出力関数は，低レベル入出力関数の上にさまざまな処理を付け加えて，より使いやすくしたものです．こちらは，FILE構造体へのポインタを用いてファイルを操作します．あるいは，ファイルでなくストリームという概念を用いているといういい方もできます．

とにかく，低レベル入出力関数でなければ実行しにくい特別な種類の操作を行うのでなければ，基本的に高レベル入出力関数を用いるべきです．高レベル入出力関数ならそれだけ仮想化されているので，OSが異なっても基本的に同じ操作が行えますから安心です．そこで，今回紹介する関数群は，すべて高レベル入出力関数です．表2を見てください．紹介する関数についての標準的なプロトタイプ宣言と基本的な処理内容です．

入出力にはどのような操作が必要でしょうか．

まずは，基本的にコンソールに割り当てられていて，必要に応じてリダイレクトされる，標準入出力ストリームに対する操作です．この機能は頻繁に使われるので，専用の関数がいくつか用意されています．getchar()，putchar()，gets()，puts()などの標準入力から文字を読み込んだり，標準出力に文字を出力したりする関数ですね．Cではこのほかに書式付きの入出力関数と呼ばれる関数群があって，数値に符号を付けたり，16進で表示したりするような機能があります．書式付きの入力関数にはprintf()やscanf()がありますが，これらは後でまとめてふれることにします．

入出力関数は，その関数の名前によってその機能がわかります．読み込みを行うのがget～関数，出力を行うのがput～関数です．文字単位で入出力を行う関数が～char関数，1行単位の文字列で入出力を行うのがstringの略で～s関数になっています．覚えやすいですね．注意しておきますが，これらの関数は標準入出力に対して操作

Cプログラミング用語
オフセット　ある基準からの相対的な位置のこと．メモリやファイルの位置を示すのに使われる．

を行います．

さて，すべての入出力操作が標準入出力に対してだけ行われるなら，以上の関数だけで話はすむのですが，そうはいきません．もうひとつ，ファイルに対する入出力を忘れるわけにはいかないでしょう．指定したファイルに対して何らかの処理をさせたいとなると，ファイルをオープンして処理を行いクローズするという，一連の操作を行う必要が出てきます．

ファイルに対する入出力は，ファイルの内容によって操作を2通りに分類できます．ひとつはテキストファイルを1文字ずつ，あるいは1行ずつ，読み書きを行うテキスト処理，もうひとつは，あるレコード単位，C言語の文法でいえば構造体単位であるとか，一定の長さのバイナリ・ブロック単位で行う操作です．

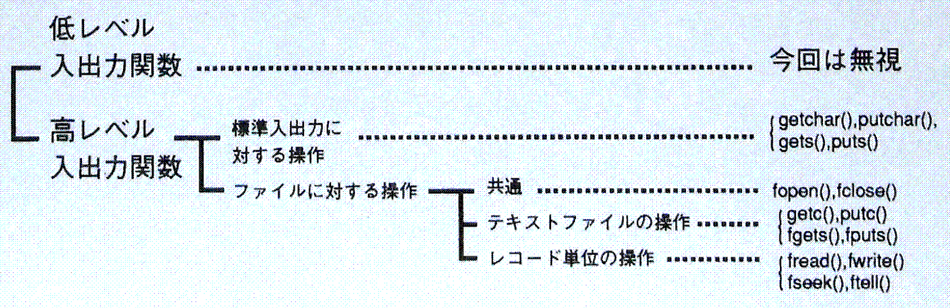
この処理には，それぞれ関数が用意されています．ただし，このときMS-DOSには特有の問題があります．Cが最初に広く使われたUNIXというOS環境では，ファイルの種類は1種類だけでした．MS-DOSも，本来のファイル管理メカニズムからいえば，1種類のファイルしかサポートしていません．この点は同じなのですが，テキストファイルの構造が，UNIXと異なっているため，UNIXのテキストファイルをエミュレートするために，テキストファイルとしてオープンするか，バイナリファイルとしてオープンするかをfopen()時に決める必要があります．UNIXでは行の区切り符合にNL(¥n)が使われていたのに対し，MS-DOSではCR(¥r)とLF(¥n)のペアが用いられているためです．プ

表2　入出力関数

以下の関数プロトタイプは<stdio.h>で宣言されている
したがって，実際の使用時には<stdio.h>をincludeすること

FILE *fopen(const char *filename, const char *mode);	ファイルをオープンする．modeの意味は本文参照．エラーの場合はNULLを返す．
int fclose(FILE *stream);	ファイルをクローズする．エラーの場合はEOFを返す．
char *fgets(char *s, int n, FILE *stream);	ストリームから1行を読み込む．ただし読み込まれる文字列の最大長は(n − 1)文字まで．文字列の最後にはターミネータ(¥0)が付加される．改行文字は文字列に含まれる．エラーの場合はNULLを返す．
int fputs(const char *s, FILE *stream);	ストリームへ1行書き出す．出力後，改行文字を付加して出力はしない．エラーの場合はEOFを返す．
char *gets(char *s);	標準入力から1行読み込む．改行文字は文字列から取り除かれる．
int puts(cosnt char *s);	標準出力へ1行書き出す．改行文字を自動的に付加する．
int getc(FILE *stream);	ストリームから1文字読み込む．エラーの場合は，EOFを返す．
int putc(int c, FILE *stream);	ストリームへ1文字出力する．エラーの場合はEOFを返す．
int getchar(void);	標準入力から1文字入力する．
int putchar(int c);	標準出力へ1文字出力する．
int fseek(FILE *stream, long offset, int whence);	ファイルポインタを指定した位置へ移動する．
long ftell(FILE *stream);	ファイルポインタの現在値を得る．
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);	ptrで指される場所へ，sizeバイトの大きさのデータをn個ストリームから読み込む．読み込まれたデータの個数を返す．エラーが発生した場合，あるいはファイルの終わりに到達した場合は，0ないし指定したnより小さい数値が返される．
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);	ptrで指される場所から，sizeバイトの大きさのデータをn個ストリームへ出力する．書き出されたデータの個数を返す．エラーが発生した場合は，0ないしnより小さい数値が返される．

図1　入出力関数の分類



型宣言　各種変数を利用するときコンパイラに対して宣言をする．宣言しなくても自動的に型を判断して変数を用意してくれる処理系もある．

ログラムの互換性を保つため，MS-DOSではテキストファイルとしてオープンされたファイルに対しては，読み書きを行うときに行の区切り符合をNLに変換しています．そのために，グラフィック用のデータなど，バイナリ形式のファイルを読み書きする場合は，符号の変換を行わないように指定しなければいけないのです．レコード単位のバイナリファイルと行単位のテキストファイル処理は，処理内容も多少性質が異なっていますが，MS-DOSの場合にはさらにこのような違いにも気を配らなければいけません．

リスト1

```
#include    <stdio.h>

FILE *fp1, *fp2;
    .
    .
    if((fp1 = fopen(input_file_name, "r") == NULL {
        ;   /* 入力ファイルがオープンできない   */
            /* エラー処理を行う             */
    }
    if((fp2 = fopen(output_file_name, "w") == NULL {
        ;   /* 出力ファイルを作成できない   */
            /* エラー処理を行う         */
    }
    while(入力ファイルの残りがある) {
        getc()やfgets()で読み込む;
        何か操作を行う;
        putc()やfputs()で出力する;
    }
    fclose(fp1);
    fclose(fp2);
}
```

テキストファイル操作

テキストファイルの操作は，プログラムを作成する上で非常によく利用するもののひとつです．多くのC言語の入門書が，テキストファイル操作の例題を載せているのではないでしょうか．

そもそも，ファイルの先頭から順に読み書きするだけですむ処理なら，すべてのデータを人が読めるようなデータ形式に変換して入出力することによって，バイナリ形式のデータファイルを使用せずに，テキストファイルの処理だけですませることができます（もっともグラフィック・データなどはファイルサイズが倍以上になるので実用的ではないかもしれません）．

プログラムを作成する上で，データをテキストファイルにしておくと，いろいろと便利なことがあります．テキストファイルなら，プログラムの出力が正しいかを確認するにも，エディタを使ったり，あるいは単にコンソールに出力するだけでできますし，テスト用のデータもエディタを使って簡単に用意できます．OS上で利用できるツール類との親和性もテキストファイルの方が優れていますから，テキストファイルかバイナリファイルか，どちらのファイル形式をとっても大差ないやと考えられる場合は，テキストファイルにしたほうがよいでしょう．MS-DOS上のC言語の処理系は，通常はテキストファイルを取り扱うだろうということで，とくにバイナリファイルであると指定しない限りは，テキストファイルとしてファイルを取り扱っています．

リスト1はテキストファイル操作の一般形式です．

●ファイルのオープン

fopen(filename, mode)の形でファイルをオープンします．modeはファイルのアクセス方法を示すもので，表3のような文字列を使います．

r はreadの，w はwrite，a はappendの頭文字です．よく使うのはrとwでしょう．末尾に+が付いていると，読み書き両方ができるようになりますが，ふつうはレコード単位で記録されているファイルをランダムアクセスするときに使われます．テキストファイルに対しては，あまり使いでがありません．

●ファイルの終りの判定

whileループの中では，入力ファイルを頭から順番に読み込みながら，何らかの処理を行います．この時，入力ファイルの残りがあるかどうか調べるには，専用のfeof(FILE *stream)という関数が存在するのですが，たいていの場合は使われません．ファイルの終りに達した場合，あるいは何かエラーが発生した場合には，getc()ならEOFという定数が返されますし，fgets()ならNULLが返されます．ですから，読み込んだデータが正常かどうを判定する際に，ファイルの終端に達したかどうかも必ずわかります．したがって，リスト1のwhile()の部分は，しばしばリスト2のように書かれています．

● getc()とgetchar()
● putc()とputchar()

リスト2ではgetc(fp)でファイルの終りを判定しています．getc()を標準入力専用にしたものがgetchar(),

Cプログラミング用語
型変換　ある型の値を別の型に変更すること．演算の際に必要となる場合もある．自動的に変換して演算を行う処理系もある．

同様にputc()を標準出力専用にしたものがputchar()です．getchar()とputchar()はgetc(), putc()と同じものと考えてよいのですが，すべてマクロとして実装されています．マクロですから，不注意な書き方をすると，処理系(コンパイラの種類)によっては正常な動作をしないことがあるかもしれません．マクロとして実装されている関数を利用する場合は，

```
putchar(++ch)
```

などのような形式で記述するのはやめたほうがいいでしょう．

表3　fopenのmode

mode	意味
"r"	ファイルを読み込み専用にオープンする．
"w"	ファイルを書き込み用に作成する． 同じファイル名のファイルが存在すると，そのファイルは消去される．
"a"	ファイルを追加書き込み用にオープンする． 既存のファイルの末尾から書き込めるように準備する．
"r+"	既存のファイルに対して読み書き両方ができるようにオープンする．
"w+"	新規のファイルを読み書き両方ができるようにオープンする．
"a+"	既存のファイルの末尾から，読み書き両方ができるようにオープンする．

リスト2

```
int ch;
    while((ch = getc(fp)) != EOF) {
        処理
    }

あるいは
char s[LBUFSIZ];
    while(fgets(s, LBUFSIZ, fp) != NULL) {
        処理
    }
```

● gets()とfgets()
● puts()とfputs()

gets()とfgets()はそれぞれ標準入力，ファイルから1行単位で入力する関数です．1行単位で入出力を行う関数は，標準入出力専用の関数とストリームの指定ができる関数とでは仕様が異なっていますから，注意してください．たとえば，gets()は，fgets()と異なり，結果を受け取る文字列の長さを指定できませんから，考えていたより長い文字列が入力された場合，文字配列の大きさを上回る領域が書き換えられて，暴走してしまう危険があります．また，gets()は入力の時に行末の改行文字が取り去られ，puts()では逆に付け加えて出力されます．単純にfgets()とgets()を交換したり，puts()とfputs()を交換したりすると，不要な改行が入ったり入らなかったりしますから注意しましょう．

これらの関数の違いをいちいち覚えるのも面倒だという人は，gets(s)の代わりにfgets(s, n, stdin), puts(s)の代わりにfputs(s, stdout)をつねに使う習慣にすればいいでしょう．gets()やputs()はデバッグ時に検査用の入出力を行うような場合に限って使うようにするのです．そうすれば，ふだんは統一して扱うことができ，トラブルを減らせると思います．

●ファイルのクローズ

使い終ったファイルはfclose()でクローズします．ファイルクローズの関数はテキストファイルでもバイナリファイルでも同じです．MS-DOSのようなシングルタスクのOSでは，万一ファイルをクローズし忘れてもそれほど大きなトラブルにはなりませんが，マルチタスクのOSでは，同じファイルを利用するほかのプログラムの妨げになる可能性もでてきます．オープンしたファイルを使い終ったら，すぐにクローズするプログラムを書くようにしましょう．

バイナリファイル，レコード単位の処理

さて，テキスト処理だけならこれまで説明した関数だけでほとんどのことは可能なのですが，そうはいかない場合もあります．ひとつは，バイナリファイルを扱う場合，もうひとつは，データベースでいうレコード単位のアクセスを行う場合です．後者の場合は，場合によってはランダムアクセスを行いながら，データを書き換える必要もでてきます．

●バイナリファイルのシーケンシャルアクセス

まずバイナリファイルをシーケンシャルに，つまり頭から順番にアクセスする場合をみてみましょう．バイナリファイルをアクセスする場合は，ファイルのオープン時に，fopen()のmode文字列の末尾に "b" を付けて，

fopen("abc.dat", "rb")

のようにするのが標準的な方法です．処理系によっては，テキストファイルの場合には"t"，バイナリファイルの場合は"b"とするものもあります．お使いの処理系のマニュアルで，fopen()について調べてください．

では，バイナリファイルをコピーするだけのプログラムを書いてみましょう．いままでに紹介した関数でも書くことはできますが，バイナリファイルには「行」という単位はありませんからfgets()やfputs()は使えません．

getc()とputc()を使ってファイルコピーを行うプログラムがリスト3（copy1.c）です．

エラー処理が繁雑になるので，まだ紹介していないexit()という関数を使いましたが，これはプログラムの実行を中断してしまうための関数です．copy1.cでもファイルのコピーはできますが，1バイトずつ読み出しては書き込んでいるため，いまいち効率がよくありません．同じ内容でコピーするだけなら，1バイトずつの処理を行う必然性はありません．まとめてごっそりと読み込んで，同じくまとめて書き出すことができれば，関数を呼び出す回数も減らすことができます．テキストファイルを「行」の単位で読み書きするのと同じように，適当な大きさのまとまり単位で読み書きすればよいのです．この適当な大きさのまとまりをとりあえずブロックと呼びましょう．ブロック単位で読み書きするための関数が，fread()とfwrite()です．これらの関数を使えば，ある大きさのブロックを何個かまとめて読み書きすることができます．そのため，この2つの関数は，それぞれ4つの引数をとります．最初の引数がブロックが格納されている，あるいは格納されるメモリを指すポインタ．次にブロックのサイズ，ブロックの個数，最後にストリームを指定するFILE構造体へのポインタです．なお，fread()，fwrite()では，size_tというデータ型が使われていますが，これはstddef.hというヘッダファイルの中で定義されている，符号なしの整数型です．

論より実例，ブロック単位の入出力に挑戦しましょう．copy1.cを書き換えて，fread()とfwrite()を使うようにしたプログラムをリスト4（copy2.c）に示します．

実行してみると，copy1よりcopy2の方が早くコピーできるでしょう．それだけ，ブロック単位で読み書きすることの効果が表われているのです．この場合，fread()のところでは1バイトのブロックが，BLKSIZ個だけ続いているようなバッファを使っていると考えて，コーディングしてあります．ですから，fread()の返す値は読み込まれたバイト数と解釈できるのです．

●バイナリファイルのランダムアクセス

これよりずっと大きなブロックをいくつか読み込むことも，構造体から構成される表を読み書きするときなどに使われます．たとえば，あるゲームのハイスコアを記録するための構造体があるとします（リスト5）．

リスト3　copy1.c

```
/* バイナリファイルのコピー 1文字単位バージョン    */

#include    <stdio.h>

int main(int argc, char *argv[]) {
int ch;
FILE *fp1, *fp2;
    if(argc != 3) {
        puts("Usage:copy1 source-file dest-file");
        exit(1);
    }
    if((fp1 = fopen(argv[1], "rb")) == NULL) {
        puts("Cannot open source-file");
        exit(1);
    }
    if((fp2 = fopen(argv[2], "wb")) == NULL) {
        puts("Cannot create dest-file");
        exit(1);
    }
    while((ch = getc(fp1)) != EOF) {
        putc(ch, fp2);
    }
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

リスト4　copy2.c

```
/* バイナリファイルのコピー ブロック単位バージョン  */

#include    <stdio.h>

#define BLKSIZ  10240

int main(int argc, char *argv[]) {
char buf[BLKSIZ];
FILE *fp1, *fp2;
size_t n;
    if(argc != 3) {
        puts("Usage:copy1 source-file dest-file");
        exit(1);
    }
    if((fp1 = fopen(argv[1], "rb")) == NULL) {
        puts("Cannot open source-file");
        exit(1);
    }
    if((fp2 = fopen(argv[2], "wb")) == NULL) {
        puts("Cannot create dest-file");
        exit(1);
    }
    while((n = fread(buf, sizeof(char), BLKSIZ, fp1)) > 0) {
        fwrite(buf, sizeof(char), n, fp2);
    }
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

Cプログラミング用語
疑似命令　コンパイル時などに，目的ファイルへ変換されるのではなく，処理系に対し指令を行う命令を差す．

リスト5

```
struct hiscore_t {
    char name[10];
    long score;
    date_t date;      /* 日付を表すデータ */
};
```

リスト6

```
struct hiscore_t hiscore_tb[10];
int n;
    fp = fopen("hiscore.dat", "rb");
    n = fread(hiscore_tb, sizeof(struct hiscore_t), 10, fp);
```

hiscore.datというファイルに，つねに上位10名の記録が残されているとしましょう．リスト6のようにすれば，ハイスコア記録をメモリに読み込むことができます．sizeof演算子は，hiscore_tのバイト数を返す演算子です．fread()は読み込んだデータの個数（この場合はハイスコア記録の数）を返します．もしもnが10ではなかったら，きっとそのゲームの実行回数は10回未満だったのでしょう．

このように，一定の大きさのデータ（レコード）が順番に並んでいるファイルの場合は，一度に全部をメモリに読み込んだり，ファイルの先頭から順番に読み書きすること以外に，ランダムアクセスという手法で任意の位置のデータをダイレクトに読み書きすることもできます．たとえばリスト6の例で，3番目のデータだけをいきなり読み込みたければ，fseek()で(sizeof(struct hiscore_t)＊3)バイト目から読みたいという指定を行えば可能になります．fseek()を使えば，任意の位置へのランダムアクセスが可能です．fseek(fp, offset, whence)の形で，whenceで示される基準位置からのバイト数によって次に読み書きする場所，ファイルポインタを変更します．whenceの意味は，

0	ファイルの先頭
1	現在値
2	ファイルの終り

です．offsetの型はlongですから，大きいファイルを操作するのに十分ですし，負の値を使うこともできます．

さらに，fopen()でファイルを開くときに，モード指定の文字列の末尾に'+'を付ければ，読み込みと書き出しを切り替えて使うこともできますから，速度を別にすれば配列のようにファイルを操作することも可能です．ただし，読み書きを切り替えるときには，ファイルポインタを移動する必要のないときでも必ずfseek()を実行しなければいけません．

このようなランダムアクセスの手法は，データベース操作のような，高度な応用に使われることが多いですから，入門レベルのプログラムではあまり必要ないでしょうが．

書式付き入出力

Cの入門書では，いの一番にprintf()が登場すると思います．それだけ基本的でよく使うということなのでしょう．でも，Cをはじめたばかりのころはフォーマット文字列の意味や使い方がよくわからないために，せっかくの便利な機能を使いこなせないことも少なくありません．

● printf()ファミリー

では，まずprintf()ファミリーから見ていきましょう．printf()，fprintf()，sprintf()が代表的なものです．printf()，fprintf()はそれぞれ標準出力，ストリーム出力に書式付きで出力するもの．sprintf()は，出力する代わりに文字配列中に結果を書き込むもので，書式付き出力というより高級文字列操作関数といった見方ができます．これらの関数は，出力時のフォーマットを指示することができます．それを指示するのが，フォーマット文字列です．フォーマット文字列の意味を表5に示します．

ざっと表を見ただけでは，実際にどう使っていいのかわかりにくいでしょうから，ここではsprintf()の使い方について，ファイル名の組み立てを例にして見てみましょう．いま，複数のデータファイルを出力するプログラムがあるとします．ファイル名はFILE0001.DAT，FILE0002.DATというように，FILEnnnn.DATのnnnnのところに4桁の番号が順番に入るように付けることにします．当然，番号を指定してファイル名を作り出す関数を書きたくなりますよね．そこで，次のような関数を考えてみました．

mk_datafn(int n, char ＊ fn)

n	番号
fn	結果を格納するバッファ

Cプログラミング用語
共用体　複数のメンバーが集合したデータ構造で，共用体の各メンバーが同じアドレスを参照する．

では，mk_datafn()の処理はどのようにしたらよいでしょうか．こんなとき，BASICで文字列の結合（+演算子）を使ってプログラムを書いていた方は，ついstrcat()と一時変数をいくつか使って組み立てようとする傾向があるように思います．この部分はsprintf()を使えば一発です．単に" FILE%d.DAT" だと，FILE1.DATとかFILE12.DATになってしまいます．" FILE%0d.DAT" というフォーマットなら，ファイル名の文字数がつねに等しくなります（リスト7）．

もうひとつ，入出力するファイルの拡張子だけを変える場合を考えてみましょう．拡張子の付いていないファイル名がfnに格納されていて，拡張子がそれぞれ" FOO" と" BAR" のファイル名を作り出すには，リスト8のようにすればよいのです．

● scanf()ファミリー

printf()が出力を行うのに対して，scanf()の方は入力のための関数です．こちらもscanf()，fscanf()，sscanf()のような種類があります．ところがこちらはどうもprintf()ほどは使いでがないようです（書式文字列に正規表現のパターンマッチが使えたりするとすごいのですが）．

さらに標準入力のscanf()は，行の概念を無視してしまうようなことがあって，多少不便でもあります．たとえば，1行あたり2つの数値が書かれた入力から，行ごとの和を求めるプログラムを作ったとします（リスト9）．

このプログラムは，誤った入力に非常に弱くなっています．たとえば，

```
1 2
3 4
5
6 7
8 x

m n
```

というような入力ファイルscanftst.datを作成しておいて，

```
scanf1 < scanftst.dat
```

とすると，数値が欠けている3行目からおかしなことになります．scanf()自身が行単位の操作が不得意なことが原因なので，fgets()とsscanf()を組み合わせて，リスト10のように改良することで，ずいぶんと安全なプログラムになります．基本的には，fgets()とsscanf()をこの例のように組み合わせて使うスタイルをお薦めします．

Learning C　文字分類関数

Cには文字種を分類するための関数やマクロがいくつか用意されています．漢字，とくにシフトJIS特有の関数については簡単に別項で触れることとして，

表4　書式付き入出力関数

以下の関数プロトタイプは<stdio.h>で宣言されている
したがって，実際の使用時には<stdio.h>をincludeすること

int printf(const char *format, ...);	書式formatに従って，標準出力へ出力を行う．
int fprintf(FILE *fp, const char *format, ...);	書式formatに従って，出力ストリームへ出力を行う．
int sprintf(char *s, const char *format, ...);	書式formatに従って，文字列を組み立ててsに書き込む．
int scanf(const char *format, ...);	書式formatに従って，標準入力から入力を行う．
int fscanf(FILE *fp, const char *format, ...);	書式formatに従って，入力ストリームから入力を行う．
int sscanf(const char *s, const char *format, ...);	書式formatに従って，文字列sから変換を行う．

リスト7

```
char *mk_datafn(int n, char *fn) {
    sprintf(fn, "FILE%04d.DAT", n);
    return fn;
}
```

リスト8

```
sprintf(fn1, "%s.FOO", fn);
sprintf(fn2, "%s.BAR", fn);
```


Cプログラミング用語
グローバル変数　プログラム中で有効となる変数で，広域変数とも呼ぶ．

表 5　printf()とscanf()の書式指定オプション

以下の書式指定は比較的よく使われるものを簡単に示したものです．詳細は処理系のマニュアルを参照してください．

● printf()の書式指定

書式指定の文字列は，'%'で始まり変換文字で終わる構文をもつ書式指定と，それ以外の文字から構成されます．書式指定以外の文字は，そのまま出力されます．

書式指定の構文　%[flags][width][.prec][h | l | L]type

frags　フラグ文字の並びで，出力の左詰め右詰めや符号を付けるかなどの指定

−	結果を左詰めで出力．
+	つねに符号をつけて数値を出力する．
' '(space)	正の数値を出力する際には＋記号を付けずに空白を置く．
#	出力のtypeがわかるように変換する． 8進フォーマットなら0を，16進フォーマットなら0xか0Xを，浮動小数点数なら小数点を，出力に付加する．

width　出力される最少文字数．ただし，出力すべき対象がこの文字数より多くなれば，そちらが優先される．0ではじまると，出力の左側余白に0が詰め込まれる．

prec　出力される文字の最大数．ただし浮動小数点数の場合は有効数字（あるいは小数点以下の桁数）であり，整数の場合は最低この数まで0を詰めて出力される．

[h | l | L]

h	値をshortかunsigned shortと解釈する．
l	値をlongかunsigned longと解釈する．
L	値をlong doubleと解釈する．

type

変換文字	引数の型	意味
d	int	符号付き10進数として出力する．
i	int	dと同じ．
o	int	符号なし8進数として出力する．
u	int	符号なし10進数として出力する．
x	int	符号なし16進数として出力する．ただしabcdefgの文字が使われる．
X	int	xと，ABCDEFGの文字が使われることを除いて同じ．
f	double	符号付き999.9999の形式で出力する．
e	double	符号付き9.999e[+	-]999の形式で出力する．
g	double	fとeのうちの適当なほうの形式で出力する．
E	double	eと，指数表示の文字がEになっている点を除いて同じ．
G	double	fとEのうちの適当なほうの形式で出力する．
c	char	文字を出力する．
s	char*	文字列を出力する．
%	――	単に%が出力される．

● scanf()の書式指定

書式指定の文字列は，'%'ではじまり変換文字で終る構文を持つ書式指定と，空白文字，さらにそれ以外の文字から構成されます．その3種類の文字によって，次のような動作が起こります．

書式指定	それに対応する引数に，入力を変換して代入する．
空白文字	空白でない入力が見つかるまで，入力を読み飛ばす．
その他	次の空白でないフィールドとマッチするはずの文字．

書式指定の構文　%[*][width][h | l | L]type

*****　この文字があると，この書式指定に対応する入力は無視される．入力動作は行うが，scanf()の引数ポインタの指す場所には代入を行わない．入力のスキップを指示する．

width　この書式指定に対応するフィールドの最大文字数を指定する．

[h | l | L]　引数ポインタが指す変数の型を表す．

h	shortかunsigned short
l	longかunsigned longまたはdouble
L	long double

type

変換文字	引数の型	意味
d	int*	10進数として入力する．
i	int*	整数として入力する．0で始まる数なら8進数として，0xか0Xで始まる数なら16進数として，それ以外は10進数として入力する．
o	int*	8進数として入力する．
u	unsigned int*	符号なし10進数として入力する．
x	int*	16進数として入力する．
f	float*	浮動小数点数として入力する．
e	float*	fと同じ．
g	float*	fと同じ．
c	char*	文字を入力する．
s	char*	文字列を入力する．

Cプログラミング用語

高級言語　CPUの構造に束縛されない言語．アセンブル言語のようにCPUよりでなく，より自然の言語に近い．

リスト9　scanf1.c

```
#include    <stdio.h>

void main(void) {
int x1, x2;
    while(scanf("%d %d", &x1, &x2) != 0) {
        printf("value: %d + %d = %d¥n", x1, x2, x1 + x2);
    }
}

```

リスト10　scanf2.c

```
#include    <stdio.h>

#define LBUFSIZ 80

void main(void) {
int x1, x2;
char buf[LBUFSIZ];
    while(fgets(buf, LBUFSIZ, stdin) != NULL) {
        if(sscanf(buf, "%d %d", &x1, &x2) != 2) {
            fprintf(stderr, "Format error: %s", buf);
        }
        else {
            printf("value: %d + %d = %d¥n", x1, x2, x1 + x2);
        }
    }
}

```

まずは基本的なASCIIコード関係ののから見てみましょう.

Cでは, 文字の比較に関しては, 文字を整数と同じように扱って比較演算子を適用することになっています. また, Cで扱う文字符号はある程度の順序性が仮定され, しかもMS-DOSレベルではASCII符号に近いものを扱っていますから, たとえばある文字chがアルファベットかどうかを調べるなら,

```
'A' <= ch && ch <= 'Z' ||
'a' <= ch && ch <= 'z'
```

と書いても正しく判定されます. しかしいつもこう書くのも繁雑ですし, 文字符号体系をつねに意識していなければならないのも面倒です. そこで, Cでは標準的に表6のようなマクロが提供されています.

表6　文字分類関数

以下の関数プロトタイプ（マクロ定義）は, <ctype.h>で宣言されている
したがって, 使用する際は<ctype.h>をincludeすること

int isalpha(int ch);	chがアルファベットか調べる. (A-Z, a-z)
int isupper(int ch);	chが大文字か調べる. (A-Z)
int islower(int ch);	chが小文字か調べる. (a-z)
int isdigit(int ch);	chが数字か調べる. (0-9)
int isxdigit(int ch);	chが16進数の数字か調べる. (0-9, A-F, a-f)
int isalnum(int ch);	chがアルファベットまたは数字か調べる. (0-9, A-Z, a-z)
int isspace(int ch);	chが空白文字か調べる. (' ', '¥t', '¥r', '¥n', '¥v', '¥f')
int ispunct(int ch);	chが区切り符合か調べる. (iscntrl(ch) || isspace(ch))
int isprint(int ch);	chが印字可能か, つまり制御文字ではないか, 調べる.
int isgraph(int ch);	chが目に見える形で印字可能か調べる. つまり' 'を取り除いたisprint(ch).
int iscntrl(int ch);	chが制御文字か調べる.
int isascii(int ch);	chが0から127の範囲に含まれるか調べる.

種類も多いし, 見慣れない名前はあるし……と, 戸惑うかもしれませんが, そのときはマニュアルを参照すればよいのです. だいたい, こんな関数があったっけ, というくらいの記憶を残しておけば十分です.

これらを使用する上では, 注意事項があります. ひとつは, これらはマクロで実装されることが多いので, 引数に副作用のある式を使うと正しく動作する保証はないということです. つまり, isalnum(ch++)などは正しくインクリメントする前のchについて判定される保証はありませんし, 呼び出した結果のchはひとつだけインクリメントされているという保証もありません.

次に, chはintと宣言されていますが, 一般にisascii()を除いて, 0から127までの値とEOFという定数値以外の値を引数にしても正しい結果が得られるかどうかもわかりません. 普通にテキストファイルを1文字ずつ読んでいれば, あまりこのような異常な値を引数にこれらのマクロを呼び出すことはないとは思いますが, なにか判定がおかしいと思えるときは確認したほうがよいでしょう.

Cプログラミング用語
構造化　プログラムの制御が上位のモジュールから下位のモジュールに渡される階層構造をとるプログラミング手法

Learning C 文字の変換関数

文字を変換するといえば，JIS漢字コードとシフトJISコードの変換のようなものを思い浮かべるかもしれませんが，2バイト系の文字コードは後まわしにします．ここでは，toupper()とtolower()の2つだけを紹介します．これらはアルファベットの大文字と小文字の間で変換を行う関数です．引数の文字がアルファベットでなければ，同じ文字を返します．つまり，toupper('a')は'A'を返しますが，toupper('*')は'*'を返します．

たとえば，プログラムの実行中に，

これでよいですか？（Y/N）

というようなメッセージを出力して確認を行うことがありますね．この時にキー入力される文字は，CAPS LOCKキーの状態などによって，大文字小文字のどちらが入力されるかわかりません．Yesの入力がされたかどうかを調べるのに，

```
ch = getchar();
if(ch == 'Y' || ch == 'y')  {
}
```

というようにやってもよいのですが，

```
ch = getchar();
if(toupper(ch) == 'Y')  {
}
```

としてしまう方法もあります．同じことが，コマンドラインの解析の時に，オプションの大文字小文字を無視する場合には使えます．大文字小文字を区別しないのであれば，1文字入力した直後にどっちかに統一してしまおう，というわけです．

表7　文字の変換関数

以下の関数プロトタイプは，<ctype.h>で宣言されている
したがって，使用する際は<ctype.h>をincludeすること

関数	説明
int toupper(int ch);	chを大文字に変換する．
int tolower(int ch);	chを小文字に変換する．

表8　文字列操作関数

次の関数プロトタイプは，<string.h>で宣言されている
したがって，使用する際は<string.h>をincludeすること

関数	説明
char *strcat(char *dest, const char *src);	destの後ろにsrcを追加する．新しい領域に結合したものを作成するのでなく，destの後ろに直接書き込むので，destには結果を格納するのに十分なサイズが必要になる．
char *strncat(char *dest, const char *src, size_t len);	destの後ろにsrcを最大len文字まで追加する．srcがlenよりも長い場合は，その後ろにある文字は追加されない．
int strcmp(const char *s1, const char *s2);	s1とs2を比較する． s1 < s2　のとき　負の値 s1 == s2　のとき　0 s1 > s2　のとき　正の値 を返す．
int strncmp(const char *s1, const char *s2, size_t len);	s1とs2を，先頭からlen文字までの範囲で比較する． それ以外はstrcmp()と同じ．
int stricmp(const char *s1, const char *s2);	s1とs2を，アルファベットの大文字小文字を無視して比較する． それ以外はstrcmp()と同じ．
char *strcpy(char *dest, const char *src);	srcの内容をdestへ転送する． 文字列を指す2つのポインタ，s1とs2があったとき， s1 = s2; では，内容が転送されずにポインタの値が変化することに注意． この関数では，ポインタの値は変化させずにポインタの指すメモリの内容が書き換えられる．
char *strncpy(char *dest, const char *src, size_t len);	srcの内容が最大len文字だけdestへ転送される． それ以外はstrcpy()と同じ．
size_t strlen(char *s);	文字列sの長さを得る．
char *strchr(const char *s, int c);	文字列sの先頭から，文字cがないか調べて，見つかったらその文字の位置を返す．cが見つからなければNULLを返す．
char *strpbrk(const char *s1, const char *s2);	文字列s1の先頭から，文字列s2に含まれる文字の内のどれかが存在しないか調べる．最初に見つかった場所を返す． 見つからなければNULLを返す．

Cプログラミング用語
構造体　必ずしも同一の型ではない複数のメンバーが集合したデータ構造．

Learning C 文字列操作関数

文字の分類を行う関数も種類が多いのですが，文字列の比較や結合，検査を行うための関数はさらに種類があります．ここでは，簡単に表にまとめるだけにしますが，基本的にすべてstr～()のような名前が付けられているので，マニュアルの中でも探しやすいと思います．また，strn～()のように，真ん中にnumberの頭文字のnが入っている関数は，先頭からn文字目までの文字を対象に操作を行うこと．stri～()というように，ignore caseの頭文字のiが入っているものは，アルファベットの大文字小文字の違いを無視すること．の2点を覚えておけば，あまり不便はないと思います．

とにかく，ここにあげたのは非常に基本的なほんの一部の関数だけです．マニュアルでstr～()のあたりを眺めてみることを薦めます．

Learning C 文字列ー数値変換関数

ここで紹介する関数は，その気になればsscanf()ですべて同じことができるのですが，使い方が簡便でリンクされるオブジェクトも小さいので，好んで使う人もいます．表9の3つを知っていれば十分でしょう．

表9　文字列ー数値変換関数

次の関数プロトタイプは，<stdlib.h>で宣言されている
したがって，使用する際は<stdlib.h>をincludeすること

int atoi(const char *s); sを数値を示す文字列と解釈して，intに変換する．
long atol(const char *s); sを数値を示す文字列と解釈して，longに変換する．
double atof(const char *s); sを数値を示す文字列と解釈して，doubleに変換する．

これらの関数は，「ASCII to 数値型」の省略形でこんな名前が付いていますから覚えやすいと思います．atofのfはfloting point，浮動小数点でしょう．それぞれの型の数値と解釈できなくなった時点で，変換をとりやめ，値を返します．最初から数値と解釈できない文字列なら，当然0が返されます．

Learning C シフトJIS関連の関数

表10　シフトJIS関連の関数

unsigned short jistojms(unsigned short ch); JISコードで表された文字chをシフトJISコードに変換する．
unsigned short jmstojis(unsigned short ch); シフトJISコードで表された文字chをJISコードに変換する．
int btom(const char *s, int nbyte); 1バイト系の文字と2バイト系の文字の混在した文字列sの先頭からnbyteバイト目までに含まれる文字数を求める． byte to mojiの略．
int mtob(const char *s, int nmoji); 1バイト系の文字と2バイト系の文字の混在した文字列sの先頭からnmoji文字目までに含まれるバイト数を求める． btom()の逆
int iskanji(int c); 文字cが，シフトJISコードの第1バイト目か判定する．
int iskanji2(int c); 文字cが，シフトJISコードの第2バイト目か判定する．

筆者はあまり日本語文字列の処理が必要なプログラムを書かないので，適切なアドバイスができるかどうかわかりませんが，この項目に入る関数は以前に比べてずいぶん増えました．jstr～()のような関数名やiskanji()などの関数で，ASCII用の(本当はASCIIコード以外の符号体系でもよいのですが）文字列操作や文字の分類関数を拡張した形を，ほとんど網羅しています．一部を除けば名前だけでだいたい想像が付きますし，細かい点について言及しようとすると文字符号の問題から説明しないと不十分な感もあります．

ここでは，日本語文字列処理でよく使われる，シフトJISと本来のJIS漢字コードとの相互変換，文字数とバイト数の対応，シフトJISの第1バイト目と第2バイト目の判別に限って紹介します．

なお，日本語文字列を扱う際には，コンパイラの設定によっては問題が生ずる可能性がありますから，お使いのCコンパイラのマニュアルのオプション設定や日本語処理に関する注意事項をチェ

Cプログラミング用語
再帰呼び出し　自分自身を呼び出す関数呼び出しである．

ックする必要があります．

Learning C
そのほかのよく使われる関数

いままで出てきた関数で，Cの入門用テキストに出てくる関数のほとんどを見たことになると思いますが，これにいくつか補足を加えたいと思います．

プログラムの強制終了

● exit()とatexit()

exit()は有名ですが，atexit()はちょっとマイナーかなとも思いますが，使うと便利なこともあるので紹介しておきます．exit()はプログラムを強制終了させるための関数です．exit(status)は，main()関数の中でreturn status;を行うのと同じ意味になりますので，main()関数の実行後に行われるいくつかの処理はきちんと実行されます．たとえば，まだオープンしたままのファイルをきちんとクローズするような処理ですね．通常のreturn文ではmain()の外側からは呼び出した場所へしか返りませんが，どの関数の中からでもそれ以降のプログラムの実行を取りやめることができます．

atexit()は，あまり使われているところを見かけませんが，非常に役に立つことがあります．たとえば，プログラムの中で割り込みベクタを変えたり，特殊なハードウェアを操作している場合，そのプログラムを終える場合にはきちんと後始末をしないと，ほかのプログラムの迷惑になったり，あるいはハードウェア・トラブルの原因になったりします．つねにそのプログラムがmain()を最後まで実行してから終了してくれるなら，main()の最後に後始末をする関数を呼び出せばよいのですが，下請けルーチンでエラーが発生してプログラムが強制終了させられるかもしれない場合には問題が残ります．そこで，後始末の必要な処理を行った直後に，後始末をする関数を登録してしまって，プログラムが終了する際にはその関数が必ず実行されるようにするのに使います．それにはリスト11のようにコーディングします．

do_it()という関数が，なにか特定の後始末をしないと危険な関数だとします．このプログラムは，func1()を2回呼び出していますが，最初の呼び出しのときにエラーが発生したら，普通にそのまま終了します．ところが，2回目の呼び出しでエラーが発生したら，終了する直前にrestore()が実行されます．atexit()がないと，エラー処理の異なる2種類の関数が必要になります．また，このように小さくて見通しのよいプログラムなら，必要性は薄いのですが，どこでexit()が起こるか把握しにくい大きなプログラムの場合や，多人数で手分けしてプログラムを作成する際にも役に立ちます．

メモリ管理関係

表11　強制終了関数

次の関数プロトタイプは，<stdlib.h>で宣言されている
したがって，使用する際は<stdlib.h>をincludeすること

void exit(int status);	プログラムを終了する．statusの値は，MS-DOSのバッチファイル内で，errorlevelとして参照できる．
int atexit(atexit_t func);	プログラムを終了する直前にfunc()を呼び出すように登録する．

リスト11

```
/* atexit()の例 */
#include    <stdlib.h>

atexit_t restore(void) {
    後始末
}

void func1(void) {
    何らかの処理
    if(error) { /* エラー発生    */
        exit(1);
    }
}

void main(void) {
    func1();    /* 1回目の呼び出し */
    do_it();    /* この関数で後始末の必要なことを行う    */
    if(atexit(restore) == 0) {
        restore();  /* atexit()の登録に失敗したら，アボート */
        exit(1);
    }
    func1();    /* 2回目の呼び出し */
}
```

Cプログラミング用語
最適化　高級言語において無駄なコードを省き，プログラムサイズを小さく，または実行速度を速くするためにプログラムを加工すること．

メモリ管理関係の関数も，Cでプログラムを書く際には必要ですね．メモリ管理といっても，メモリの割り当てと解放だけですが．ここではmalloc()とfree()だけを紹介します．

この2つの関数だけで，十分でしょう．Cで大きな表などを扱う場合は，一般に表を指すポインタを宣言しておいて，malloc()でメモリを割り当てます．一時的に必要な表では，必要なくなったときにfree()で解放すれば，メモリを有効に使えます．実例は次のqsort()と一緒にリスト12(dirsort.c)に示します．

表12　メモリ管理関数

次の関数プロトタイプは，<stdlib.h>で宣言されている
したがって，使用する際は<stdlib.h>をincludeすること

void　*malloc(size_t size); sizeバイトのメモリを確保して，そのメモリブロックへのポインタを返す．メモリが確保できなければ，NULLを返す．
void free(void　*block); 割り当てられたメモリブロックを開放する．

表操作

次に，qsort()とbsearch()，lsearch()という代表的な表操作用の関数を紹介します．Cで作成するアプリケーションの中には，データを主記憶上の表に登録して，検索を行いながら，何らかの処理を行うという場合も多いと思います．表の検索用のルーチンは，自前で用意してもたいした量にはならないので，ついついライブラリを利用しないことも多いのですが，存在だけでも知っておいて損はないでしょう．

qsort()は，あの有名なクイックソート・アルゴリズムを用いたソート，並べ変えを行うための関数です．引数が多くて覚えにくいのですが，使うときにはマニュアルを見ればよいのです．引数の中で，比較を行う関数は一般にはちょっと面倒です．fcmp(x, y)で，

```
*x <  *yのとき　負の値
*x == *yのとき　0
*x >  *yのとき　正の値
```

を返すような関数です．表の要素が文字列だけなら，strcmp()などがそのまま使えます．整数なら，(*x - *y)の値を返すような関数を定義すればよいのです．表の要素が構造体になっている場合は，ソートのキーになる要素を取り出して比較した結果を返す関数を定義する必要があります．

表13　表操作用の関数

次の関数プロトタイプは，<stdlib.h>で宣言されている
したがって，使用する際は<stdlib.h>をincludeすること

void qsort(void　*base, size_t nelem, size_t width, int (fcmp)(const void　*, const void　*)); baseで指されたnelem個の要素を持つ配列をソートする． 一つの要素のサイズはwidthバイトである． fcmp()は要素の大小比較を行う関数
void　*bsearch(const void　*key, const void　*base, size_t nelem, size_t width, int (fcmp)(const void　*, const void　*)); baseで指されたnelem個の要素の中からkeyを持つ要素を検索して，その要素へのポインタを返す． 1つの要素のサイズはwidthバイトである． fcmp()は要素の大小比較を行う関数 keyと一致する要素が見つからなければNULLを返す．
void　*lsearch(const void　*key, const void　*base, size_t　*num, size_t width, int (fcmp)(const void　*, const void　*)); baseで指された*num個の要素の中からkeyを持つ要素を検索して，その要素へのポインタを返す． 1つの要素のサイズはwidthバイトである． fcmp()は要素の大小比較を行う関数 keyと一致する要素が見つからなければNULLを返す．

qsort()がわかってしまえば，bsearch()やlsearch()も似たような引数を取りますから，すぐ使えると思います．しかし，注意すべきこともいくつかあります．bsearch()で検索可能な表は，2分割探索法を使うので，検索に用いるキーであらかじめソートされていなければならないこと．同じくbsearch()で，検索キーと同じ要素がいくつかあった場合は，どの要素が返されるかは不定なこと．つまり，

```
bar
foo
foo
foobar
```

という表で，fooを検索した場合，検索結果として2番目の要素が返されるか3番目が返されるかわかりません．bsearch()を用いる表は，一般には2重登録しないようにチェックしたほうが安全だと思います．

この点，lsearch()は先頭から順番に探していくアルゴリズムなので，表をソートしておく必要はないし，同じキーの要素が複数あった場合は先頭に近いほうが確実に返されます．ただし，検索速度はbsearch()に比べて遅く

Cプログラミング用語
シフトJIS　JIS漢字コードをシフトし，グラフィック文字部に割り当てたコード体系．MS-DOSや一部のUNIXで採用されている．

なりますし，lsearch()は簡単なループですぐに書けてしまうので，マニュアルをひっくり返して引数を調べる前に自分で直接書いてしまえることも欠点かもしれませんね．また，要素数を指定する引数が，qsort()とbsearch()では要素数を直接書けばよいのに対し，lsearch()では要素数を示すポインタである必要があります．コンパイラの型チェックに対する警告には注意したほうがよいかもしれません．

では，実例として，ディスクのディレクトリをファイル名でソートして出力するプログラム，dirsort.cをリスト12に示します．使い方は

```
dir | dirsort
```

という形式のみが許されます．このプログラムには実用的な価値は全然なくて，読者にはあまりに申し訳ないプログラムですが，例題ということで大目にみてください．ディレクトリを読み込む表を確保するのにmalloc()を，ファイル名の大小比較としてstrncmp()も，併用しています．

環境変数を利用する関数

最後に環境変数へのアクセス法を覚えておきましょう．MS-CやQuickCを使用している方は，コンパイラのオプションを環境変数で指定するのは普通にやっているでしょうし，多くのソフトウェアがTMPのような環境変数をテンポラリ・ファイルを作成するディレクトリとして参照しているのはご存じでしょう．これからあなたの作成するプログラムも，ほとんど変更する必要のないオプションやディレクトリの指定などは環境変数を使っても指定できるようにすると便利だと思います．これにはgetenv()を用います．

つまり，TMPという環境変数の内容を知りたければ，getenv('TMP')とします．環境変数へのアクセス法はほかにもあるのですが，これだけ覚えておけば十分でしょう．では，例として，SAMPLEという環境変数の内容を表示するプログラムenvsampl.c（リスト13）と，TMPという環境変数で示されるディレクトリの下にsample.tmpというファイルを作成するためのパス名を作成して表示するプログラムtmptest.c（リスト14）を見てください．後者は，いままでに出てきた関数をいくつか使っています．

リスト12　dirsort.c

```
/*      dirsort.c
    ディレクトリをファイル名でソートして出力する。
    必ず、dir | dirsort
    の形式で利用すること。それ以外の利用法は、たぶんない。
*/

#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <ctype.h>

#define DLINSIZ 45      /* dirの出力の１行当りの文字数の最大値（より大きい） */
#define CMPLEN  12      /* ファイル名は行の先頭から12文字まで */
#define MAXELEM 1000    /* 扱える行数   */

#define LBUFSIZ 80

char *dirtb[MAXELEM];

/* 要素間の比較を行う関数。                                         */
/*  この関数の引数はdirtbの要素を指すポインタなので、引数は        */
/*  実際には文字へのポインタのポインタになっていることに注意。     */

int lcmp(const void *s1, const void *s2) {
    return strncmp(*(const char **)s1, *(const char **)s2, CMPLEN);
}

void main(void) {
int i;
int n;
char buf[LBUFSIZ];
    for(i = 0; i < MAXELEM; i++) {
        if((dirtb[i] = malloc(DLINSIZ*sizeof(char))) == NULL) {
                        /* メモリ割り当ての失敗 */
            fprintf(stderr, "Need more memory.¥n");
            exit(1);
        }
    }

    for(n = 0; fgets(buf, LBUFSIZ, stdin) != NULL; ) {
                        /* 不要な行を取り去る    */
        if(isalpha(buf[0])) {
                        /* 先頭文字がアルファベットなら、確かにファイル名   */
            strncpy(dirtb[n++], buf, DLINSIZ);
        }
    }
                        /* ソートされるのは、この場合ポインタの配列である   */
                        /* ことに注意。したがって、第３引数はDLINSIZではない*/
    qsort(dirtb, n, sizeof(char *), lcmp);
                        /* 結果出力 */
    for(i = 0; i < n; i++) {
        fputs(dirtb[i], stdout);
    }
                        /* メモリの解放 */
    for(i = 0; i < MAXELEM; i++) {
        free(dirtb[i]);
    }
}
```

Cプログラミング用語
条件コンパイル　ヘッダ指令を使用して，コンパイルする部分や，しない部分を作ること．

Learning C 最後に

Cのライブラリ関数を上手に使いこなす秘訣にはどのようなものがあるのでしょうか．

私がもっともお薦めしたいのは，仕事の暇なときに，ライブラリ関数のマニュアルをひと通り読むことです．え？　何百もあるライブラリ関数を暗記するのかって，驚かないでください．そんなに細かく読んだり，全部覚えようと気合を入れて読む必要はありません．そんな読み方でなく，こたつにもぐりこんで，寝転がりながら読むような感じで（勤務時間中ならばまじめそうな顔でいるべきでしょうが）気楽に目を通すのです．で，だいたいこんな処理はライブラリで用意されているのか，くらいの覚え方をしてください．次になにかプログラムを組んでいる最中に，「あ，そういえばこんな関数を見たことがあったような気がする」という程度に頭の隅で引っ掛かるようになればしめたものです．実際の関数名や使い方は，それからマニュアルを開くなり，関係のありそうなヘッダファイルを読んで検索しオンライン・マニュアルで調べるなりすればよいのですから．

この点から考えれば，MS-CやQuickCなどは，ライブラリ関数の説明はオンライン・マニュアル主体になってきてしまって，製本したマニュアルには詳しいことは載せないようになってきてしまっているのは残念です．オンライン・マニュアルは関数名がわかっていて，使い方だけがわからないときに利用するものです．自分の行いたいことがライブラリに用意されているかどうかは，すぐにはわかりません．

Cを使いはじめてしばらくは，今回紹介した関数だけでかなりのことができると思います．が，自分の使っているCにはどんなライブラリ関数が用意されているのか，そうやってマニュアルを眺めるのも楽しいと思います．たぶん大半は「あーこんな関数，当分使わないな」というものばかりだと思いますが，それで十分ではないでしょうか．

表14　環境変数を利用する関数

次の関数プロトタイプは，<stdlib.h>で宣言されている
したがって，使用する際は<stdlib.h>をincludeすること

```
char ＊ getenv(const char ＊ name);
        nameで指定される環境変数の内容を示すポインタを返す．
        指定された環境変数がなければNULLを返す．
```

リスト13　envsampl.c

```
/*            envsampl.c
        単に環境変数SAMPLEの内容を表示する
*/
#include    <stdio.h>
#include    <stdlib.h>

void main(void) {
char *s;
    s = getenv("SAMPLE");
    if(s == NULL) {
        puts("SAMPLEという環境変数が存在しません");
    }
    else {
        puts(s);
    }
}

```

リスト14　tmptest.c

```
/*      tmptest.c
    環境変数TMPの下に、sample.tmpというファイルを作成するための
    パス名を合成して表示する。
*/

#include    <stdio.h>
#include    <stdlib.h>

#define PATHLEN 100
#define FNAME   "sample.tmp"

void main(void) {
char path[PATHLEN];
char *tmp;
    if((tmp = getenv("TMP")) == NULL) {
                    /* TMPがなければカレントディレクトリ    */
        strcpy(path, FNAME);
    }
    else {
        if(tmp[strlen(tmp) - 1] == '¥¥') {
                    /* TMPの末尾に¥が付いていれば、そのまま連結する */
            sprintf(path, "%s%s", tmp, FNAME);
        }
        else {
                    /* TMPの末尾に¥が付いていないなら、¥を挿入する  */
            sprintf(path, "%s¥¥%s", tmp, FNAME);
        }
    }
    puts(path);
}

```


Cプログラミング用語
スタック　データの記憶方法のひとつ．データをスタック領域に退避し，要求があった場合最後に退避したデータから取り出す．

特集 Cの学び方・教え方

Teaching C

企業内技術者教育にみるC言語の「現在」

Cを教えるその前に

春は新入社員の季節である．ここでは，新入社員に対するCの教え方を取り上げよう．最近では，まったくの未経験者がソフトウェア要員として入社することも少なくない．Cを教える以前に，コンピュータに関するごく基本的な知識からはじめなければならない場合もある．プロのソフトウェア技術者を育てるためのコンピュータ教育には，どういった内容が求められているのか，即戦力につながるC言語の教え方はあるのか，企業ではどのような視点で新入社員教育を行っているのかについてのレポートをまじえながら，新入社員に対するC言語の教え方のポイントを探ってみたい．

編集部

Teaching C
企業内コンピュータ教育の現実

プログラマやシステムエンジニアといったソフトウェア技術者は，人材の不足が叫ばれて久しい．情報処理を専門に学んできた学生を集めるだけでは人手不足に追いつかず，まったく知識のないほかの専門分野の出身者に対しても独自に教育を行って，ソフトウェア技術者として育て上げるという会社も増えている．ゼロからスタートして，短期間でプログラムが組めるようになるまでには，どのような教育を行っているのだろうか．まずは会社内での新入社員教育の実際から見ていきたい．

初心者には コンピュータの基礎から

日本電気グループのひとつである日本電気コンピュータシステム㈱は，汎用コンピュータからエンジニアリングワークステーション，パソコンにいたるまで，ハードウェア・ソフトウェアのトータルシステムを開発，販売している会社である．パソコンの分野では，MS-Windows対応の通信ソフト「WINTERM」の開発・販売元として知られている．同社では理工系および若干名の情報処理系の専門学校の卒業生をのぞいて，新入社員の大半はコンピュータに触れた経験がほとんどない，いわばまったくの素人ばかり．社内での教育と，OJTで1人前のシステムエンジニアやシステムプログラマに育て上げている．

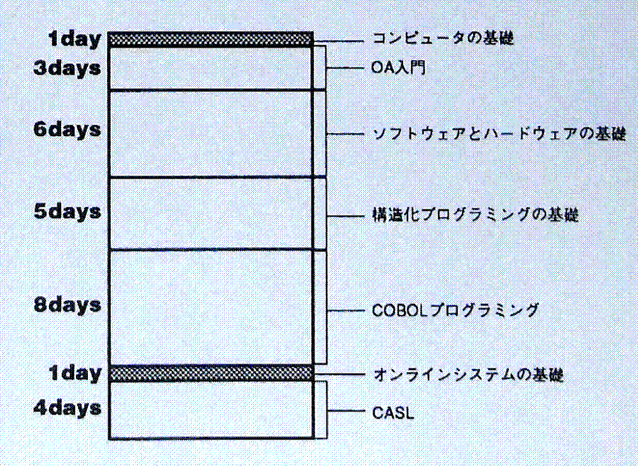
管理本部教育部教育課長の井上文明氏は，新入社員に対する教育のレベルについて次のように語っている．「新人教育は，本来仕様書を見て簡単なプログラムが作れる程度，自分で簡単なデバッグができる程度までいけばよいと考えています．コンピュータの基礎教育のコースを終えた時点では，とりあえずCOBOLで簡単なプログラムが組めるようになること，さらにUNIXのコースをおえた時点では，C言語で簡単なプログラムが組めて，UNIXのツールを使ってデバッグができるようになることが目標です．ただし，最近ではだんだんその内容も豊富になってきて，現在では情報処理第2種の試験に出題される問題程度の初歩的な知識を身につけるところまでが含まれていますね」．

同社での新入社員向けカリキュラムでは，まず30日間かけて，コンピュータの基礎を学び，その後配属された部署によって機種別，OS別の研修を受ける．「30日間の基礎コースの内訳ですが，コンピュータの入門ということで1日，OA入門が3日．OA入門ではキーボードに慣れてもらって，NECのLANシリーズというOAソフトのシリーズの操作方法を覚えます．次にソフトウェアとハードウェアの基礎を6日間，構造化プログラミングの基礎を5日間，COBOLプログラミングが実習を含めて8日間，オンラインシステムの基礎が1日，最後にCASLを4日間という日程で行います」（井上氏）．

最初にプログラミング言語としてCOBOLを学ぶのは，その後にオンラインシステムの基礎という項目がきているのを見てもわかるように，多くの新入社員がその後アプリケーションシステムのシステムエンジニアとして業務に携わることからきているのだそうだ．

CASLというのは，情報処理技術者試験の問題で使用される仮想CPU，COMETのためのアセンブラ言語である．もちろんCOMETというCPUは現実には存在しないので，研修で学んだからといって仕事の上で直接役にたつというものではない．それでもあえて，カリキュラムの中にCASLを取

図1　日本電気コンピュータシステムの基礎教育のカリキュラム



Cプログラミング用語
ストリーム　プログラムとファイルの間のデータ入出力の流れを結ぶのがストリーム．

り入れているのは，コンピュータのハードウェア，ソフトウェアの基本的機能/動作をしっかり理解してもらうためだ．

コンピュータ経験の有無でクラス分け

コンピュータの基礎的な仕組みやプログラミングについては各人の知識のレベルにはかなりの開きがある．大勢の社員を一斉に教育するとなると，問題となるのが標準となる知識レベルをどのあたりに想定するかということ．日本電気コンピュータシステムでは，約100名の新人をコンピュータの経験者とまったくの未経験者にクラス分けすることで対応している．しかし，教える内容は同じである．経験者といっても，実務で要求するレベルまで達している者はほとんどいないからだというのがその理由である．

この際に気をつけなければならないのは，新入社員の心理的な面への配慮であるという．中には経験者・未経験者という区別を，教育の内容で差別されていると誤解する新入社員もいるので，クラスを分ける際には，スタートの時点が違っても教育期間が終わるまでに同じレベルまで理解すればよいのだと必ず説明しておく．

Cのコースは4日間

新入社員は基礎教育を終えた後，配属された部署によってさらにOS別の教育プログラムを受講することになっている．同社の場合，日本電気製のコンピュータだけでなく，米国Bull HN社製品も取り扱っているため，実際に配属先の部署で担当するコンピュータの種類は多岐に渡っている．おもなOSはNECの汎用機のOSであるACOS-2，ACOS-4，ITOS，それにUNIXであるが，Bull HN社製品にはGCOS-6, GCOS-8などのOSがある．パソコンを扱う部署もあるのだが，新入社員向けのコースにはとくにOS-2やMS-DOSのコースは設けられていない．

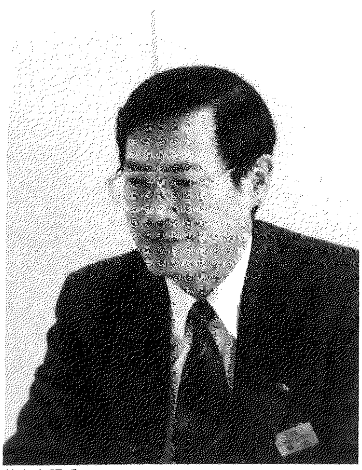
OS別のコースは，実際にコンピュータを使った実習が中心となる．UNIXの基礎教育は，全新入社員が受けるが，4日間のC言語入門コースを含むUNIX専門コースを受けるのは20名前後であるという．

実習には，PC-9800シリーズのパソコン上でPC-UXを使用している．

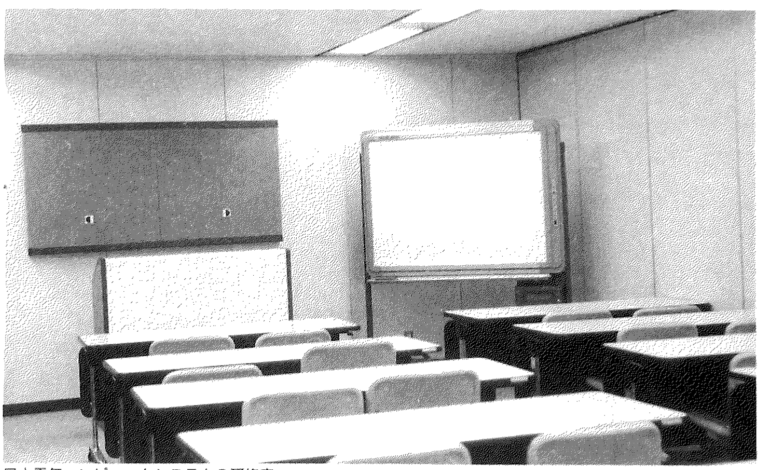
「C言語というのは，OSを知っているのと知らないのとでは，ずいぶん理解のしやすさが違います．ほかの言語と比べた場合，Cはデータの構造を表すことができますが，OSやコンピュータの仕組みを知らないと，その構造が何に使うものなのかということはピンとこないでしょうね．新人に教える場合には，それをどうやって理解させるかということが，難しいところです」（井上氏）．

コンピュータの基礎教育で，COBOLを8日間もやった後で，UNIXのコースに入って新たにCを覚えるというカリキュラムは，ことUNIXやC専門の技術者を育てるのに最適とはいえない．それは，井上氏自身も感じているという．100名近い新入社員をに短期間で効率よく教えなければならないという制約のもとでは，しかたのないことなのだろう．

C言語の入門がわずか4日間しかないというのも少々意外な感じがするが，本来NECグループの共通のカリキュラムではCのコースは3日間しかなく，さすがにそれでは時間が足りないので同社では1日増やしたのだそうだ．「C言語入門の次は本来ならばC言語の応



井上文明氏



日本電気コンピュータシステムの研修室

Cプログラミング用語
静的変数　グローバル変数などプログラムのメモリロード時に確保され，終了時までアドレスが変化することのないデータ領域

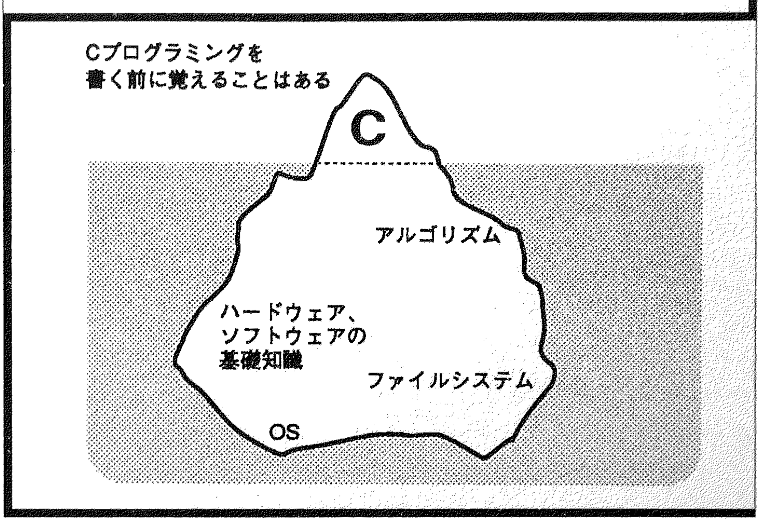
用ということになるわけですが，そこまでは新人教育の中ではちょっと無理ですし，それ以降はやはり仕事の中で覚えてもらうことになります」(井上氏)．

新入社員向けの研修でできることは，時間的にも，内容的にも制限が大きい．その限られた条件の下でどこまで何を教えればよいのか，これは教育を担当する者なら誰でも考えることだ．日本電気コンピュータシステムの新入社員教育は，まったくの未経験者をプロのソフトウェア技術者への入り口まで導いていくためのプロセスとして位置づけられている．C言語のプログラミング教育はその中の一部を占めているにすぎない．プログラミングの技術や詳しいコーディングの方法を教えることも，即仕事に活かせるという面でのメリットは大きいが，ソフトウェアを仕事とするプロの技術者を育てるためには，プログラミング以前にソフトウェア・ハードウェアについての基礎固めを行うことの重要性を見落としてはならない．

Teaching C
C言語の前に必要なもの

新入社員のコンピュータに対する知識がゼロである場合には，C言語のプログラミングの前にある程度コンピュータに関する初歩的な知識から教えるという方法は，やり方次第でかなり有効である．では，具体的にどのような知識が有効になるのか，次はそれを検討してみることにしよう．

図2　C言語を学ぶ前に必要な知識



先にアセンブラを教える

C言語はもともとOSを記述するために設計された言語である．したがって，高級言語でありながら，コンピュータのハードウェアに近い部分の記述が可能であるということは，本誌の読者なら十分ご存知だろう．

ところが，Cをはじめて学ぼうとする人には，ビット単位の操作，レジスタという概念などはなかなかつかみにくい．これは，BASICのようなほかの言語をすでに知っていても同じだ．これをクリアする方法として，Cより先にアセンブラを教えてしまうという手がある．アセンブラを学ぶためには，まずCPUの構造を知らなければならない．どのようなレジスタがあって，データの内容を変更するには，どのような命令を実行しなければならないかが明確になる．アドレス，メモリ，レジスタといった概念を教えるのにこれほど絶好の手段はない．

日本電気コンピュータシステムの場合も，CASLを基礎教育のカリキュラムに取り入れていたが，新入社員がその後どんなCPU，どのようなOSを使うのかがあらかじめわかっていれば，CASLのような仮想的なものを使う必要はない．MS-DOSやOS/2を使うCプログラマを育てるのなら，8086のCPUの仕組みとアセンブラの基本的な命令を教えればよいのだ．一度覚えた知識は今後も使えることになる．

ハード，ソフトの知識は情報処理2種で

C言語を覚える上で，OSというほど高度でなくてもよいから，とりあえずコンピュータのハードウェア，ソフトウェアに関する基礎知識はあったほうがよい．とはいえ，いまさらコンピュータにはメモリとCPUと入出力装置があって……などとはじめるのでは，教える側もたいへんだ．こういった基礎的な話がコンパクトにまとめられているのが情報処理第2種の参考書である．

情報処理技術者試験そのものは，パソコンのプログラマやUNIXのシステムプログラマよりも，事務処理などのアプリケーションシステム開発者向

Cプログラミング用語
セグメント　80x86プロセッサ独特のメモリを管理するための単位

けの色あいが濃いが，情報処理第2種の試験は，「コンピュータの仕組み」「補助記憶装置」「入出力装置」「コンピュータによる問題処理」「オペレーティングシステムの機能」など，ハードウェア・ソフトウェアの基礎的なことがらばかりである．豊富な問題集から練習問題を援用したり，新入社員の自習用に利用したりといった使い方もできる．

なお，従来情報処理試験では，プログラム作成問題でCOBOL，FORTRAN，PL/I，CASLの4言語を採用していたが，近々これにCが追加される予定で，現在前提となるCのJIS規格の制定が急ピッチで進んでいるという話もある．今後Cを教える上での情報処理試験の利用価値はいっそう高まっていくだろう．

アルゴリズムとフローチャート

C言語の場合，覚えなければならない制御構文の数は決して多くない．if文，switch文のような条件文．for文，do while文のようなループを示す文，あとはいくつかの演算子を知っていればそう不自由はしない．だからといってCがプログラミングしやすい言語だということにはならない．初心者に対する練習問題を出してみればわかる．「aとbのデータを入れ換える」くらいはなんとかなっても，「文字列を逆順に表示する」とまずお手上げになってしまう．プログラムを書くには，まずそのアルゴリズムを明確にする必要があること，アルゴリズムを記述する手段としてのフローチャートについては，あらかじめ教えておく必要があるだろう．

Teaching C

おわりに Cを教えることの意味

あらゆる問題に対処できるCプログラミングの書き方を「教える」ことがとうてい不可能である以上，どんな教え方がもっともふさわしいかは，教える相手の現在持っているコンピュータについての知識のレベル（まったくの初心者なのか，ある程度プログラミングの経験があるのか），これから業務上こなしていかなければならない分野（アプリケーション，ハードウェア制御，OS）によって，異なってくるはずだ．より適切な「C言語の教え方」を考える上で，本稿の内容がわずかばかりのヒントにでもなれば幸いである．

COLUMN

Cを教える人に捧げる　初心者のココには要注意！

■「BASIC経験者はGOTO文が好き」

Cの前にBASICやFORTRANなどのほかのプログラミング言語を知っていると，逆にとまどってしまうことが多い．何も知らなければ，すんなりと頭に入ってくるようなことでも，これまで身につけた知識がかえって邪魔をする．とくにBASIC経験者はすぐGOTO文を書きたがるので要注意である．

■「数学が苦手な人は「関数」に弱い」

関数と聞いただけであとはもうひたすら拒否反応を示してしまうことがある．とくに学生時代数学が大の苦手だったという人に多い．こういうときは，「C言語の関数というのは，せいぜい『機能』という程度のものだ」とやさしく説明して，まず拒否反応をとりのぞいてあげることが必要だ．

■「汎用機ユーザーはシステム資源に鈍感」

たとえワープロソフトしか使ったことがないとしても，パソコンのユーザーはメモリやディスクの容量に敏感で，システム設定を自分で行うことにも抵抗がない．逆に汎用機の端末しか使ったことのない人は，メモリやディスクに対しての感覚が鈍いし，MAKEファイルを作ったり，環境を自分で設定したりすることを面倒だと感じるようだ．

■「エラーメッセージがわからない」

Cコンパイラの出すエラーメッセージは一般的に不親切である．これには，ほかのプログラミング言語でコンパイルしたことのある人ほどこたえる．まずはC言語がOSを記述する言語として誕生したという経緯を話して，プロのソフトウェア技術者ともなれば，この程度のメッセージで内容は理解できるようになると説明しよう．やがてはエラーメッセージとはそういうものだと思えるようになるのだから．

■「フリーフォーマットでは書けない」

ラベルや行番号を決められた位置に記述する言語に慣れた人は，カラム位置を意識しすぎるあまりCのプログラムへのとっかかりがつかめなくなってしまうことがある．

■「ポインタ・構造体・配列で誰でもつまづく」

どんな初心者もたいていここでひっかかる．これを一発で教える秘策はまだあみだされていない．「わからない」を連発する初心者には，この3つは関係が深いので，単独でどれかひとつだけわかるというものではないが，ある程度プログラムを作っていけばどこかで必ずわかるようになるとアドバイスを．

Cプログラミング用語
宣言文　変数の各種型や関数のためにメモリエリアを確保するために記述する．

特集 Cの学び方・教え方

Teaching C

基本は重要なことを少しだけ教えること

C言語セミナーの教え方

新しく入社してきた新人を一人前のCプログラマに育てる，あるいはほかの言語から新たにCを学び直す，どちらもじつに頭の痛い問題である．ここでは，Microsoft Universityの講師としてC言語のセミナーを担当しておられる栗田洋一氏，荻野俊哉氏の両氏に，Cを教える人間の立場から，Cの教え方のコツとノウハウについて語っていただいた．

富士ソフトウエア株式会社教育出版部品川教育センター
栗田洋一氏・荻野俊哉氏に聞く

Teaching Microsoft University 「C言語入門」「C言語応用」

――まず富士ソフトウエアで行っているセミナーの内容についてお聞かせください．

栗田：弊社では現在，Windows, OS/2，UNIX，Intel CPUなど，54コースのセミナーを8教室150台のマシンで実施しております．C言語としては，「C言語入門」，「C言語応用」の2コースを実施しております．基本的には，どちらのセミナーもC言語は初めてという方にC言語を理解していただくという形のものです．

担当は，荻野が入門，私が応用をおもに受け持っています．

「C言語入門」では，変数など基本的なものから関数の作り方まで，「C言語応用」のほうでは，C言語で一番難しいといわれるポインタと構造体に重点を置いて教えています．したがって，「C言語入門」と「C言語応用」の2コースを受講して，ひと通りのことが理解できるということになります．

――セミナーの受講者としては，どの程度のレベルの方を対象にしているの

Cプログラミング用語
ソースファイル　プログラム本体を保持する単位で，ソースファイルを各処理系にかけることにより機械語の目的ファイルが作られる．

でしょうか？
荻野：入門に関しては，まったくプログラミングの知識のない初心者を対象にしています．

――まったくの初心者となると，Ｃの文法を教える以前に教えなければいけないことがかなりあると思うのですが．
荻野：いま入門編ではQuickCを使っているのですが，この処理系はプログラムを書いてメニューから「実行」というコマンドをマウスでクリックするだけで，コンパイル・リンク・実行という動作を自動的に行ってくれます．MS-DOSのことなどはなにも考えなくてもよいので，とくにMS-DOSの知識は必要ありません．
　また，開発手順や用語についても，ひと通り説明いたしますので，問題ありません．

Teaching C
セミナーの進め方
まず，プログラムを動かすこと

――セミナーでは，どのような形で講義を進めていかれるのですか．
荻野：入門のほうは，まず最初に，とにかくプログラムを作って動かしてみようという感じで進めていきます．プログラムを動かすことから始めて，変数とは何か，関数分け，関数の作り方と解説していくわけです．
　２日間のセミナーの間，つねにプログラムの構造化設計というものを意識して講義を行っています．
　また，プログラムを作る際には，ただ動けばいいのではなく，きちんと美しいプログラムを書くようにしましょうと教えています．コメントやマクロ定義についても，その重要性を指導しています．
栗田：入門・応用，どちらのセミナーも最後に総合的な演習問題があり，その演習問題の課題プログラム作成を目標として講義を進めていくようになっています．一口にＣ言語といってもかなり範囲が広いですから，最後の演習問題を作るため，という形でポイントを絞って教えています．
荻野：実際のセミナーでは，最終日に半日かけて，課題プログラムを受講者に作ってもらうわけです．

――ライブラリ関数の使用方法などについて，とくに処理系によって異なる関数などについては，どのようにお教えになるのでしょうか．
荻野：まず第一に，そういうものが存在しているということを教えます．関数の概念を教えるときに，こういう関数はすでにできあがっているので，自分で作る必要はないと．あとは，ほかにもこのような関数がありますという紹介をします．〝部品化〟についての説明ですね．
栗田：基本的にANSIに準拠した形で講義を行っていますので，とくに処理系による差というようなことについては話しておりません．そのような細かい話よりも，Ｃ言語そのものに親しんでいただくようにしています．

――初心者が，つまづきやすいポイントのようなものはあるのでしょうか．
荻野：いちがいにはいえませんが，初心者がつまづくのは，まず関数とはなにかがわからない．さらに，その関数の作り方がわからない．また，関数を呼び出した側と呼ばれた側で値を渡したり返したりする，このときの文法的な問題が理解し難いようです．
栗田：やはり，私もまず関数でつまづくと思います．その次に，モジュールごとにコンパイルし，リンクして実行形を作るという，いわゆる分割コンパイルがわからないという方も多いです．Ｃ言語ではごく一般的な作り方なんですが，こういう作り方をしたことのない方がけっこういるようですね．つねに，すべての関数を１つのファイルで作っていたという方は，そのへんでまごつくようです．
荻野：むしろほかの言語の経験がいっさいない方のほうが，こういうもんだと思って素直に受け取れるようですね．逆にずっとBASICをやっていたというような方のほうが，凝り固まったというか，とくに文法的に理解できない，素直に受け取れないといったケースがあります．

Teaching C
ポインタの教え方に秘策はない!?
メモリマップを示しながらソースをトレース

――「Ｃ言語応用」ではポインタと構造体に重点をおいて解説しているということですが，ポインタや構造体を理解させるためにどのような講義を行っているのですか．
栗田：講義で使用しているテキストでは，かなりポイントをしぼって解説しています．まず，ほかの高級言語をやってきた方ですと，アドレスという概念がないものですから，そういった基本的な話から入ります．たとえば何か変数をとったら，それにもアドレスが付けられているんですよ，というところから始めるわけです．

Ｃプログラミング用語
双岐選択文　制御文の一種で，if～else文のように条件の成立・非成立がそれぞれ独立して実行される文．

ポインタだけを説明するとかなりの方がわかってくれるようですが，なにせセミナーの期間が短いものですから，次々に新しいことがでてきて，結局消化しきれなくなってしまう方もいるようです．「C言語応用」は3日間のうち，1日半でポインタと構造体を解説して，そのあと午後からリスト処理として構造体とポインタをからめた話がでてきてしまいますから．

ポインタが理解できて，構造体が理解できて，ここまではいいんですが，このあと複合問題になるとかなり悩んでしまう方がでてきますね．

しかし，複合問題で苦労しているうちに，おのずとこれらが身に付いているはずです．

やはり，短期間にパッとわからせるような秘策というようなものはとくにありません．なるべく詳しく説明するしかないですね．たとえばメモリなど，どこになにが入っているか，動きを全部詳細に1つ1つソースに合わせて解説していくしかないです．デバッガで追いかけていくように，ソースを1行ずつトレースしていくわけですね．

荻野：基本的には，ポインタを勉強するというのではなく，たとえばリスト処理のような形でポインタを使わなければできない処理を実現してみることです．文法的な問題などは，あとで納得するということですね．

机上で全部理解してからプログラムを作るというのはまずありえないことで，実際にプログラムが動いてからこれはこうなっているから動いているんだ，という形で納得するほうが現実的だと思います．

栗田：講義で説明して，その後すぐ演習問題をやっていただくのですが，実際にやってもらうと何をやったらよいかわからない方が多いです．ですから聞いて理解できても，実際にはできないことが多いのです．それなら最初にとにかく作ってしまって，動くことを確認してからあとで文法を理解してもらうほうがかえって効果的だと思います．

Teaching

テキスト作りは課題プログラムから
プログラムに興味を持たせることが肝心

――実際に動くプログラムを作らせることが重要だということですが，具体的にどのようなプログラムを作らせるのがよいのでしょうか．

荻野：やはり，最初は誰もが興味を持てるようなゲームなどがいいと思うのですが，セミナーでは画面などに凝るのはちょっと問題があります．使用するコンピュータの機種に依存するようなものは，なるべく避けています．エスケープシーケンスをガンガン使うようなものは避けて，標準入出力だけを用いるようなシンプルなものにしています．

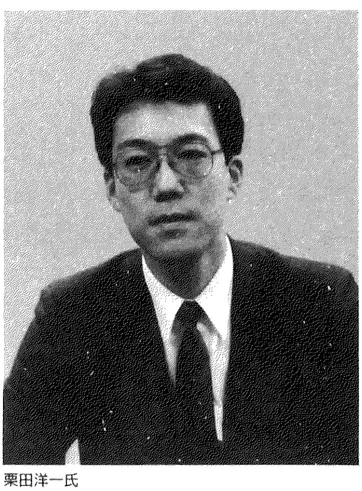
栗田：実際，「C言語入門」では，簡単な数当てゲームを演習問題にしています．

いかに受講者の興味を引く演習問題を用意するかが，セミナーを成功させる鍵になりますから，幣社のセミナーで使っているテキストを作る際には，まずどのような演習問題を作るかということから始めて，次にそれに合わせて文法的な解説部分を作っていきます．

荻野：テキストの作り方としては難しくなってしまうと思いますけれども，C言語の入門書などの最初のほうで，変数とは……などとやられても，興味のわきようがないと思います．具体的にプログラムを作ってから，こうするためにはこういうことが必要ですよ，と教えていくほうが初心者にはわかりやすいと思います．

ほかの言語である程度経験があれば，文法を照らし合わせて身に付けることもできるとは思いますが，まったくの初心者の場合には変数という概念すらわからないわけですから，なんでそんなものが必要なのかということをプログラムを作ることを通して身に付けさせていくべきでしょう．

栗田：セミナーを実施する場合には，最初に目標となるプログラムを見せておいて，こういうのを最終的に作ります，そのためには……，という形でやっていくとわりと興味をもってやれる


栗田洋一氏


荻野俊哉氏

Cプログラミング用語
多岐選択文　制御文の一種で，switch〜文のように複数の条件に対し，それぞれ実行される文．

んじゃないかと思います．

入門も応用も，各章の演習で最終的に作るプログラムの部品になるようなものを作らせるようになっていますから，イメージとしても１つの目標に向かってプログラムを作っていっているというふうになるわけです．

こうすると，セミナーが終った時点で一通りの満足感を味わってもらえますし，あとで見直す時にも自分で作ったものですからわかりやすいだろうと考えています．

また，セミナー全体を通して，プログラムの設計方法や部品化などのやり方もひと通りわかるようにしています．

Teaching
Cをどこまで教えるか？
リスト処理を理解できればポインタと構造体は十分

――C言語の学習に関して，教える側には「どこまで教えればよいのか？」，学ぶ側には「どこまで学べばよいのか？」という問題があると思うのですが．

栗田：まず最初に，Cについて必ずしもすべてのことを知らなくていいということを理解していただきたいと思います．結構知らなくてすむことも多いんですよ．

はじめのうちはあまり細かいことは教えないで，最小限必要なものだけを教えるべきです．基本さえマスターしてしまえば，自分で資料を読んで理解することができるわけですから．また，そのレベルにするのが，うちのセミナーの１つの目的なわけです．

初心者が，多すぎる情報に振り回されないようにしてあげるということが大事です．「C言語応用」では，この点を考慮してできる限りシンプルな構成にしています．ちょっと簡単すぎるかなと思える程です．

早くCを覚える，覚えさせるコツは，情報量をなるべく少なくしてあげることです．最小限の重要なものだけを教えてあげるということです．

その情報をもとに，実際にプログラムを組んでもらって，確実に知識を身に付けてもらっています．

荻野：Cにおいてまずマスターすべきことというと，「C言語入門」で教えている関数のインターフェースと関数分け――モジュール化，構造化――です．関数分けがきちんとできるかどうかというのが，入門のポイントになります．

制御文というのは，実をいうとあまりネックではないのです．何度か使ってみれば理解できるはずですし，結局は慣れの問題で，ほかの高級言語にも同じようなものがありますから．

応用では，やはりポインタと構造体ですね．

最終的にどれだけの知識が身に付けばいいかというと，リスト処理を通してポインタと構造体が理解できれば十分でしょう．リスト処理――構造体とポインタがうまくからんでいるようなもの――を含む演習問題が解けるようになれば，C言語らしいプログラミングスタイルが身につくでしょう．あとは，どんなプログラムでも応用させて自分で作れるようになります．これ以上難しい使い方というのは，そうはないですからね．

栗田：あとは，特殊な使い方ですから．普通の使い方ならば，リスト処理までで十分でしょう．

短い期間でC言語を教えるためには，やはり教える側がストーリーをきちんと作って，最小限の情報を１つ１つ身に付けていけるように，プログラムの部品を作らせていって，最終的に全体の処理を理解できるという構成が必要ですね．

――最後に，読者のなかにはこれから富士ソフトウエアのセミナーを受講しようという方もいらっしゃると思いますので，なにかアドバイスがあればお願いします．

栗田：現在，C言語では16台のマシンを用意して，定員16名でセミナーを実施しています．コースの設定として月・火曜日が「C言語入門」，水・木・金曜日が「C言語応用」というケースが多いのですが，これを連続して受けるのはちょっと苦しいと思います．消化不良を起こさないためにも，「C言語入門」を受けてから少し間をあけて，自分自身でプログラムを作るなどして教わったことを確認してから「C言語応用」を受けたほうが効果的でしょう．

受講者は，プログラミング経験の長い方から本当の初心者の方まで幅広く，プログラマ以外にも高等学校の先生や個人で受講される方もいらっしゃいます．最近の傾向としては，MS-Windowsのプログラミングを行うためにC言語のセミナーを受けに来る方が増えているようです．

１週間自分で本を読んでもたいした効果は期待できないと思いますが，セミナーを受講していただければ，最小限覚えなければいけないことが何であるのかもわかりますし，これからやっていけるという自信もつきます．

構造体やポインタなどをひと通り学んだけれどもどうもよくわからないという人には，「C言語応用」は最適だと思います．

（敬称略）

C プログラミング用語
単岐選択文　制御文の一種で，if～文のように条件が成立する時のみ実行される文．

特集 Cの学び方・教え方

Learning C

2大C言語処理系互換ライブラリの作成

Borland C++、Microsoft C 6.0の互換性

大野元久

Borland C++ 2.0とMicrosoft C 6.0は，MS-DOS上の2大C言語処理系として広く知られている．この2つの処理系はそれぞれに便利な関数や機能をもっているが，残念ながら便利な関数や機能にかぎって互換性が存在しない．そこで，これら2つの処理系の違いを明らかにするとともに，処理系間の差異を吸収する互換ライブラリの作成を試みる．

はじめに

Borland C++ 2.0（BC++）とMicrosoft C 6.0（MS-C）は，ともにANSI Cの規格に準拠したコンパイラです．どちらも準拠度が完璧だとはいえませんが，ANSI Cの範囲だけでプログラミングすればソースレベルでの互換性はほぼ保てます．

しかし，通常のアプリケーションプログラミングがそれだけで実現できることはまずありません．テキスト，グラフィックス，マウス，割り込みなどANSI Cだけでは実現できないさまざまな要求があります．どちらのコンパイラもそれらの要求に応えるライブラリ関数を用意していますが，残念ながらこれらの関数にはほとんど互換性がありません．また，処理系独自に拡張された機能もあります．BC++はTurbo C++以前と比べて比較的MS-Cとの互換性に配慮していますが，無視できない違いはいくつもあります．

この章では，PC-9801用の両コンパイラを比較し，処理系に依存しないプログラムの書き方の解説や差異を吸収するためのライブラリの作成を行うことにします．

オブジェクトレベルでの互換性

両方のコンパイラがまったく同様のオブジェクトファイルを作成するならば，コンパイルしたオブジェクトはどちらでリンクすることもできるはずです．実際，両者が使うセグメント名，メモリモデルは非常に似ています．しかし，オブジェクトレベルでの互換性の障害となる理由はいくつもあります．

まず，コンパイラ自身が使用するヘルパルーチンの名前が異なります．どちらも8086をターゲットとしたコンパイラですが，8086はコンパイラが必要とする命令をすべて備えているわけではありません．たとえば，8086にはlong（4バイト）型の乗算命令はありませんから，long型の乗算が必要になればそのためのサブルーチンを呼び出してこれを実行します．このようにCPUがサポートしていない命令を肩代りする関数をヘルパルーチンと呼びます．ヘルパルーチンはライブラリファイルに含まれていますが，名前や仕様が両者の間で異なるためヘルパルーチンが必要なオブジェクトはつねに対応するライブラリを使ってリンクされなければなりません．コンパイラによってヘルパルーチンが必要な条件は違いますが，longの乗除算・シフト，構造体の代入，far/hugeポインタの演算，スタックチェックなどでヘルパルーチンが使われます．

関数の戻り値の返し方も異なります．BC++では浮動小数値をコプロセッサ（あるいはエミュレータ）の内部レジスタを使って返しますが，MS-Cは固定領域を使います．構造体を返す場合も，BC++では呼び出し側が領域を確保しますがMS-Cでは呼び出された側が領域を確保します．

同一シンボルで異なる値が定義されている場合もあります．文字を分類するis...マクロでは両方のコンパイラとも_ctypeという文字型の配列を使っていますが，この配列の内容がそれぞれのコンパイラで異なるためis...マクロを使ったプログラムは，別の処理系のライブラリが使われると正常に動作しなくなります．fcntl.hで定義されている，O_RDONLYなどのシンボルの値もBC++とMS-Cでは違うため，これらを使うopenやcreatなどの関数に直接数値を指定してしまうと両者で移植性がなくなります．

内部で隠れて利用されているグローバル変数などもそれぞれの処理系用に作られています．BC++のグラフィックスをサポートするBGIは，BC++のスタートアップルーチンに用意されている変数を利用しますので，マイクロソフト系の処理系では利用できません．

こうして考えてみると，BC++とMS-Cで共通に使えるオブジェクトやライブラリを作成することはむずかしいように思えます．実際，Cで作成されたモジュールで共通ライブラリを作成するのは至難の業でしょう．しかし，アセンブラで独立したモジュールを作成すればこうした問題を気にすることはなくなります．ポインタや整数型の受け渡しについてはどちらの処理系でも同じですし，ヒュージモデル以外ではセグメント名やクラス名は同じです．

リスト1，2に，BC++とMS-Cの両方で使えるプログラム例と使用例を

示します．

プログラム上の違い

データ型

両者がサポートする基本型はchar, short, int, long, float, double, long, ポインタ（near/far）で，その内容はどちらも同じです．整数型は2の補数表現を使い，浮動小数型はIEEE形式になっています．

構造体において，BC++はデフォルトでバイト境界，MS-Cはワード境界になっています．たとえば，次のような構造体を作ってしまうとBC++では3バイトにおさまるようコンパイルされますが，MS-Cではメンバbdataとwdataの間に境界あわせのための空間が追加され4バイトになります．

```
struct frame  {
    char bdata;
    int wdata;
} ;
```

BC++では-aオプションで，MS-Cでは/Zp1オプションでデフォルトの境界指定を変更できますし，#pragmaを使うこともできます．しかし，できればどちらのオプションでコンパイルしてもかまわないよう，あらかじめダミーのメンバを入れておくほうがよりよいでしょう．この例の場合は，次

リスト1　getcode.asm

```
;
;       getcode:
;               PC-9801 のノーマルモードで、テキスト VRAM の内容を
;               文字コードとして取得する関数
;
.MODEL SMALL, C
TVSEG   SEGMENT AT 0A000H
TVSEG   ENDS

.CODE
;
;       unsigned getcode(int xpos, int ypos);
;               xpos = 0..79
;               ypos = 0..24
;
PUBLIC  getcode
getcode PROC    xpos:WORD, ypos:WORD
        mov     ax, ypos
        mov     cx, 160
        mul     cx
        mov     bx, xpos
        shl     bx, 1
        add     bx, ax
        mov     ax, TVSEG
        mov     es, ax
        mov     ax, es:[bx]
        cmp     ah, 0
        je      getcode_quit
;
; VRAM CODE to SHIFT-JIS
;
        dec     ax
        xchg    ah, al
        shr     ah, 1
        jnc     getcode_1
        add     al, 94
getcode_1:
        add     al, 20h
        js      getcode_2
        dec     al
getcode_2:
        add     ah, 81h
        cmp     ah, 9Fh
        jbe     getcode_quit
        add     ah, 0E0h - 0A0h

getcode_quit:
        ret
getcode ENDP
END
```

リスト2　getctest.c

```
#include <stdio.h>

unsigned getcode(int xpos, int ypos);

void main()
{
    int i;
    unsigned code;

    for (i = 0; i < 80; i++) {
        code = getcode(i, 0);
        if (code & 0xff00) {
            /* 漢字コードの時は上位バイトも出力し、*/
            /* 1桁右にずらしておく */
            putchar(code >> 8);
            i++;
        }
        putchar((unsigned char)code);
    }
}
```


Cプログラミング用語
低水準関数　ストリームを介さない，ファイルの入出力などハードウェアに密接した関数を指す．できが悪い関数を指すわけではない．

のようにします．

```
struct frame {
    char bdata;
    char _dummy;
    int wdata;
};
```

なお，構造体で使われるビットフィールドメンバは，MS-Cではlong型（32ビット）まで許されますが，BC++ではint型（16ビット）までしか許されません．16ビットの範囲では，データの並び方はどちらも同じです．

ポインタ修飾子

MS-Cは，バージョン3.0でfar/nearという80x86に特有の修飾子を考え出しました．このほかに，さまざまな修飾子がそれぞれの処理系でサポートされています．それぞれの処理系がサポートしている修飾子を比べてみましょう（表1）．

far, near, hugeなど以前から存在した修飾子は__がついていませんが，ANSIの推奨によってどちらも__がつく形式(_far, _nearなど)をサポートしています．今回の説明では使いません．また，ここで比較した内容はあくまでも目安でしかありません．同じ修飾子でも処理系によってその意味が少しずつ違います．

まず，データポインタについて説明します．far/nearについてはどちらの処理系もほとんど同じですが，少しだけ違いがあります．BC++ではfarポインタどうしの減算はlong型になりますが，MS-Cではint型になります．hugeポインタどうしの減算はどちらもlong型になります．

表1　修飾子の比較

データポインタ修飾子		コードポインタ（関数）修飾子	
BC++	MS-C	BC++	MS-C
far	far	far	far
near	near	near	near
huge	huge	huge	
_seg	_segment	interrupt	interrupt
	_based	cdecl	cdecl
_cs		pascal	pascal
_ds		(pascal)	fortran
_es		_export	_export
_ss		_loadds	_loadds
		_saveregs	_saveregs
			_fastcall

リスト3

```
/* スモールモデルでコンパイルしてください */
void test(char far *ptr)
{
  if (ptr == (char far *)NULL)
    puts("ptr is NULL");
  else
    puts("ptr is not NULL");
}
```

NULLの定義についても注意しておくほうがよいでしょう．BC++では，メモリモデルに応じてNULLは0か0Lとして定義されていますが，MS-Cでは(void *)0として定義されています．もし，BC++でNULLを(void *)0と定義してしまうとリスト3のプログラムは期待どおりに動作しなくなります．

BC++では，いったんnearポインタに変換されてしまうと，それをfarポインタに変換するときには必ずセグメントにデータセグメントの値が付加されるためです．これに対し，MS-Cでは定数を変換している限り元の値が0であればデータセグメントも0になります．これは，(char far *)(char near *)0という値を表示してみるとわかります．ポインタ変数を介して間接的に型変換を行った場合は，その変数の内容が0であるかどうかを前もって知ることはできないのでデータセグメントの値が付加されます．

hugeポインタについては，両者の間で決定的な違いがあります．BC++では，hugeポインタに対する演算のたびにオフセット値が0～15の間におさまるようセグメント／オフセットの正規化が行われます．これに対し，MS-Cではオフセットが65535を超えてはじめてセグメントの値が更新されます．このため，BC++では効率は悪いのですがhugeポインタによって任意のアドレスから任意の大きさの型に自由にアクセスすることができます．MS-Cでは限られた条件でのみセグメントの調整が行われるため効率はよいのですが，オフセットがちょうどで割り切れない場合は正しいアクセスが行われなくなります．

例をあげてみましょう．BC++では，64KBを超えるヒープメモリはfarmallocやfarcallocを使って確保

Cプログラミング用語
テキストモード　ファイルに対して文書などのテキストを読み書きする．0x0d,0x0aを改行し，0x1aをファイルの終端と判断する．

します．これらはmalloc／callocのfarヒープ対応版で実際にメモリを確保してみるとオフセットは必ず0x0004となっていることがわかります．

これに対しMS-Cではhallocを使いますが，型の大きさをあらわす第2引数が2のべき乗になっていれば戻り値のオフセットは0に，そうでなければ64KBを超える時点でちょうどオフセットが0になるように戻り値のオフセットが計算されます．hallocには，型の大きさが2のべき乗にならないときは128KB以上のメモリを確保できないという制限がありますが，まさにこの調整が有効なのは最初のセグメントを超えるところだけだからです．

たとえば，long double型は大きさが10バイトですからhalloc(10000L，sizeof(long double));では65536÷10=6553余り6となりますから，この余りの6があらかじめオフセットとして返されます．実際に使用する型と違う値を渡してしまうと，この調整がうまく行われません．たとえば，この例と同じ意味でhalloc(100000L，1);を使ってしまうとオフセットは0で返されますから，6554番目の要素が必要になるとxxxx：FFFAHから10バイトをアクセスしようとし，セグメントがうまく更新されないので正しい結果が得られません．

この問題は，配列を定義する場合にも影響します．静的データとして配列を定義する場合，すべての配列要素はセグメントにまたがることができません．したがって，2のべき乗でない大きさの型では128KB以上の大きさの配列を確保することができません．

hugeポインタに関するMS-Cの仕様は，DOS（リアルモード）では単に効率の問題でしかありません．しかし，WindowsやOS/2などのプロテクトモードではセグメントとオフセットが単に4ビットずらした加算では表現されないため，この方法でしかhugeポインタを扱えないのです．実際，Windowsアプリケーションの開発をサポートしているBC++では-hオプションで高速hugeポインタと呼ばれるMS-Cと同様のhugeポインタをサポートしています．BC++でWindowsアプリケーションを開発する場合は，自動的にこのオプションが有効になります．このオプションはDOSでも使えますが，効率をよくしようとして安易に高速hugeポインタを使おうとしてはいけません．BC++のfarmalloc/farcallocは，MS-Cのhallocのようにオフセットの調整をしてくれないためです．

BC++のセグメントポインタ（_seg）と特殊なデータポインタ（_cs，_ds，_es，_ss），MS-Cの_basedポインタはいずれも互換性がありません．記述上の移植性に配慮する限りこれらは使わないほうがよいでしょう．効率が重視されてどうしても必要な場合は，条件コンパイルを利用しどちらの形式でも書いておくべきです．

関数のための修飾子では注意すべき点があります．BC++では次のどちらの形式でも受け入れられますが，MS-Cでは後者の形式はエラーになります．

```
void (far * fptr)();
void far (* fptr)();
```

MS-Cでは修飾子はアスタリスクの直前になければなりませんが，BC++では型の後ろに記述することもできるのです．

far/nearはどちらの処理系も同じです．関数に対するhugeは，BC++でのみ有効です．BC++では，MS-Cと異なりヒュージモデルではモジュールごとにデータセグメントが作られるため，データアクセスの効率を上げるため関数の入り口部分でデータセグメントが再ロードされます．つまり，huge関数はfar呼び出しとデータセグメントの再ロードが行われる関数を意味します．MS-Cでは，ヒュージモデルはポインタがすべてヒュージポインタとして扱われる以外はラージモデルとほとんど同じで，ライブラリもラージモデルと共通のものを使います．データセグメントが変化することはないのでhuge関数はありません．MS-Cで，BC++のhuge関数と同じ効果を持たせるためにはfar _loaddsで修飾します．

MS-CのFORTRANはPascalと同じなので，あまり使う必要があるとは思えませんが必要ならばPascalに置き換えればよいでしょう．

interruptに関しては，MS-Cのマニュアルでは後ろにfarをつけるように記載されていますがなくてもよいようです．BC++では，farがあってもなくてもかまいません．また，BC++では次のどの形式も受け入れますが，MS-Cでは最初の形式しか受け付けません．BC++のマニュアルでは2番目の形式で記述されていますが，移植性に配慮する限り最初の形式で記述するようにしてください．

```
void (interrupt far * handler)();
void interrupt (far * handler)();
void interrupt far (* handler)();
```

しかし，interruptについてはもっと大きな問題があります．interruptが指定された関数は，その入り口と出口ですべてのレジスタを退避／復帰す

Cプログラミング用語
デバイス　機構または装置のこと．MS-DOSではRAMディスクや日本語入力FEPなどの仮想デバイスとして認識されることができる．

リスト 4

```
ＢＣ＋＋の場合
void interrupt far handler(unsigned bp, unsigned di, unsigned si,
        unsigned ds, unsigned es, unsigned dx, unsigned cx,
        unsigned bx, unsigned ax, unsigned ip, unsigned cs,
        unsigned flags, ...);
ＭＳ－Ｃの場合
void interrupt far handler(unsigned es, unsigned ds, unsigned di,
        unsigned si, unsigned bp, unsigned sp, unsigned bx,
        unsigned dx, unsigned cx, unsigned ax, unsigned ip,
        unsigned cs, unsigned flags);
```

るのですが，関数の引数として渡されるレジスタの順序が異なるのです（リスト4）．

これを吸収するには，引数リストをマクロで定義しておきレジスタの渡される順序に依存しないようにしておくとよいでしょう．

_exportは，WindowsやOS/2(MS-Cのみ)でエクスポート関数を定義する場合に使います．farで定義された関数をエクスポート可能関数にしようとするとただのfar関数でもよけいなプロローグコードが生成されて効率が悪いので，_exportをつけた関数だけに特別なプロローグコードを生成させるためのものです．Windowsプログラムを開発する場合，これらは同じように機能します．

_loadds, _saveregsは，それぞれデータセグメントの再ロードとすべてのレジスタの退避／復帰を指定するオプションで両者とも同じように機能します．

_fastcallについては，MS-Cで効率のよい関数呼び出しを実現するためPascal形式の呼び出しとレジスタによる引数渡しを指定するためのものです．BC++のバージョン3.0ではサポートされていますが2.0にはありません．どうしても高速化を図りたいならばPascalに置き換えておけばよいでしょう．

メモリモデル

メモリモデルについて，両者ともタイニィ／スモール／ミディアム／コンパクト／ラージ／ヒュージというメモリモデルをサポートしています．使われるコードポインタ／データポインタは，それぞれ表2のようになります．

MS-Cでは，どのメモリモデルでもDS=SSとなります．BC++では，ラージデータモデル（コンパクト／ラージ／ヒュージ）ではDS≠SSです．BC++で，DS=SSを想定するには-Fsオプションを指定します．スモールデータモデル（タイニィ／スモール／ミディアム）でDS≠SSを想定するには-mt!，-ms!，-mm!オプションを指定します．MS-Cで，DS≠SSを想定するには/Auオプションを使います．

静的データを確保する場合，BC++ではヒュージモデル以外ではつねにDGROUPに確保されます．ヒュージモデルでは“ファイル名DATA”というセグメント内に確保されます．MS-Cでは，スモールデータモデルではDGROUPに確保されます．ラージデータモデルではしきい値を設定し，それより小さなデータはDGROUPに，大きなデータは新しいセグメントに割り当てます．ヒュージモデルでは，64KBを超えるデータでも自動的にセグメントが作られます．BC++でも-Ffオプションでしきい値を設定することができますが，64KB以上を超えるデータには適用されません．このため，ヒュージモデルでも1モジュール内のデータの大きさはやはり64KBに限られます．BC++で64KBを超える静的データを確保するためには明示的にhuge修飾子を使います．

(例)

```
char huge array[1024][200];
```

この指定方法はどちらの処理系でも利用できますし，タイニィ以外であればどのメモリモデルでも使えます．MS-Cではヒュージモデルを使えばよいのですが，ヒュージモデルではすべてのポインタがhugeポインタとして扱われるためかなり効率が悪くなります．むしろ，通常はラージモデルを選択し，64KB以上を扱う必要がある

表 2　コードポインタ／データポインタの比較

	BC＋＋	MS－C
タイニィ	near／near	near／near
スモール	near／near	near／near
ミディアム	far／near	far／near
コンパクト	near／far	near／far
ラージ	far／far	far／far
ヒュージ	huge／far	far／huge

Cプログラミング用語
デバッグ　プログラムの誤りを訂正する作業．デバッガを使わず，目でプログラムの流れを追いながらする作業をハンドデバッグという．

場合にhuge修飾子を使うほうが効率的です。

さて，64KBを超える配列の大きさを取得するときは，BC++では若干問題があります．BC++の場合は，明示的にhugeを指定したものでない限りsizeofの結果はunsigned型（size_t)です．huge指定をするとsizeofの結果はunsigned long型になりますが正しい値が返されません（注：BC++ 3.0では正しい値が返されます)．MS-Cの場合は，前述したしきい値によって結果がunsigned型かunsigned long型のどちらかになります．しかし，どちらになったかを判断するためにsizeof sizeof arrayのようにするとつねに2が返されます．sizeofの結果を(unsigned long)でキャストするのが無難でしょう．

もっとも，バッファのような初期化されていない領域を静的データとして確保しておくというのはプログラミング手法としてはあまり感心しません．また，この方法はWindowsではめんどうな問題をまねきます．malloc系の関数を使って動的にメモリを確保するようにしたほうがメモリの再利用の面から考えてもよいでしょう．

80x86へのアクセス

両方とも80x86CPUをターゲットにしたコンパイラです．80x86へのアクセスを容易にするため前述のポインタ修飾子以外の拡張も行われています．

BC++では，80x86のレジスタと直接互換性を保つための疑似レジスタ変数をサポートしています．これは，_AXのようにレジスタ名の前に_をつけてあらわします．レジスタは，コンパイラが生成するコードによりいつでも破壊されることがあるわけですが，小さなBIOSルーチンを呼び出したりするにはたいへん便利な機能です．しかし，これはMS-Cでは使えません．MS-Cでレジスタに直接アクセスする場合は，インラインアセンブラを使わなければなりません．

インラインアセンブラは，どちらの処理系もサポートしています．BC++ではasm/_asmの両方が使えますが，MS-Cでは_asmしか使えませんから，移植性に配慮するならば_asmを使うべきです．また，両者とも{}でインライン部分をブロック指定することができますが，BC++の場合ははじまりの‘{’は_asmと同じ行になければなりません．BC++では，以前のバージョンからasm文は行末まででひと区切りとなっているためです．_asm文の行に‘{’がないと次の行に‘{’があってもそれは通常のCの文として認識されてしまいます．

インラインアセンブラで注意すべきなのは，MS-Cではアセンブリ文中の‘；’がコメントを意味するのに対し，BC++では文の終りをあらわすことです．BC++では，‘；’によって_asmブロックの中でも複数のアセンブリ命令を記述できますが，MS-Cで同じことをするには_asmブロックの中にさらに_asmを記述しなければなりません．ところが，BC++では_asmブロックの中に_asmを記述することを認めていないので，こうした記述はエラーになります．

通常は，命令ごとに改行するようにすれば問題ありませんが，マクロ定義で_asmブロックを記述しようとすると問題になります．マクロでは¥を使って行を連結すると改行が入らなくなってしまうからです．BC++とMS-Cでは同一行に複数のアセンブリ命令を記述する同じ方法がないため，マクロ定義で複数のアセンブル命令を定義する場合は両方の形式を定義しておかなければならないでしょう．少なくともコメントには，/*～*/の形式を使うようにしてください．

また，直接命令コードを埋め込むためにBC++では__emit__，MS-Cでは_emitをサポートしています．BC++の__emit__は組み込み関数になっていて_asm文の外で使います．MS-Cの_emitは，_asm文の中でコードを埋め込むために使います．移植性に配慮するならば，これらは使わないほうがよいでしょう．

#pragma

#pragmaは，コンパイラ固有の指令を受け入れるためにANSIで規格化されたものです．両者とも，多くの#pragma指令をサポートしていますがこれらの間にはまったく互換性がありません．よく使われそうな#pragma指令について比較してみましょう

表3　#pragma指令の比較

	BC++	MS−C
スタックチェックの無効化	#pragma option -N-	#pragma check_stack(off)
スタックチェックの有効化	#pragma option -N	#pragma check_stack(on)
構造体のワード境界指定	#pragma option -a	#pragma pack(2)
構造体のバイト境界指定	#pragma option -a-	#pragma pack(1)

Cプログラミング用語
日本語処理　全角文字は2バイトで表現されるため，日本語文字列操作では特別な関数に操作や判断が必要な場合がある．

（表3）．

なお，BC++では，スタックチェックはデフォルトでは無効ですが，MS-Cでは有効になっています．また，構造体についてはBC++のデフォルトはバイト境界ですが，MS-Cではワード境界です．

#pragmaそのものは，処理系が認識できなければ無視するのがANSI Cの仕様ですから，どちらかでしか使えない#pragmaを指定しても基本的には他の処理系で動作します．しかし，当然ですがその#pragmaの機能を期待して作成したプログラムは別の処理系では正しく動作しなくなる可能性があります．

ライブラリの違い

さて，もっとも違いが顕著なのが提供されているライブラリ関数でしょう．ANSIで規格化されているものはともかく，それ以外の関数については独自に，しかも多様な拡張がなされています．同じような機能でも違う名前になっていたり，もっと問題なのは同じ名前なのに機能が違うものもあることです．前者はその名前で新しい関数を用意すればよいのですが，後者はどちらかに合わせた関数を作るか差し障りのない範囲でそのまま使わなければなりません．

個々の関数について，今回作成する互換ライブラリに関する内容を含めて詳しく見てみましょう．

【コンソール入出力関数】

よく問題になるのが，コンソール入出力関数です．getchやputchなどは，MS-CではDOSの低レベルファンクションを使っているだけですが，BC++ではPC-9801のテキストVRAMに直接書き込むようになっています．このため，MS-Cでこれらの関数を使っても他機種で動作させることができますが，BC++の場合は完全にPC-9801依存のプログラムとなります．MS-Cでは，グラフィックスライブラリの一部として直接テキストVRAMにアクセスする関数を用意していますが，これはこれでコンソール入出力関数とはまったく異なるものです．

また，1文字入力するためのgetchについても，MS-CではDOSファンクションの08Hを使っていますがBC++では07Hを使っています．したがって，MS-Cのgetchは[CTRL]-[C]で中断されますが，BC++のgetchは[CTRL]-[C]で中断されません．ただし，入力バッファの状態を調べるkbhitではどちらもDOSファンクションの0BHを使っているため，この関数を呼び出すと[CTRL]-[C]で中断できてしまいます．

【タイムゾーン】

時間を扱う関数では，PC-9801用BC++ではタイムゾーンが日本時間になっているのに対して，MS-Cではパシフィックタイムゾーンになっていて夏時間まで計算されるようになっています（MS-Cもバージョン5.1では日本時間になっていました）．したがって，両者で同じ時間にtime関数を呼び出すと，この時間差だけ返される値がずれることになります．どちらでも同じ値が返されることを期待するならば，プログラムの先頭でputenv("TZ=JST-9");としておきタイムゾーンをあらわす環境変数TZを日本時間にあわせておくべきでしょう．

【farポインタを扱うマクロ】

dos.hではfarポインタのセグメントとオフセットを取得するためのFP_SEG／FP_OFFというマクロが定義されていますが，これもBC++とMS-Cの間では異なります．BC++では，引数がポインタでも配列名でもかまわないのですが，MS-Cでは必ずfarポインタでなければなりません．これに対し，BC++ではMK_FPという別のマクロで新たなfarポインタを作成できます．

今回は，MS-Cの定義を無効にし，BC++形式に合わせるようにしました．これにより次のような書式は使えなくなります．

```
FP_SEG(farptr) = 0xABCD;
```

しかし，あるポインタのセグメント部やオフセット部だけを変更することはまずないでしょう．MS-Cでfarポインタを作成する次のような書き方は，

```
FP_SEG(farptr) = _psp;
FP_OFF(farptr) = 0;
```

BC++の定義では次のように記述します．

```
farptr = MK_FP(_psp, 0);
```

BC++形式であればfarポインタだけでなくnearポインタや配列名，変数のアドレスについてもに正しくセグメント／オフセットを得られるようになります．実際，互換ライブラリの中でもBC++形式の定義を多用しています．

【構造体REGS】

同じくdos.hでは，REGSという

intdosやint86で使われるレジスタのための構造体が定義されています．この構造体は，BC++ではflagsというメンバが用意されていて割り込み発行後のすべてのフラグの内容がわかるようになっています．MS-Cではcflagというメンバでキャリーフラグを判断することしかできません．あまり数は多くありませんがゼロフラグによって処理の結果を判断するものがあります．この場合，MS-Cではインラインアセンブラを使わなければならないことになります．

【strerror】

errnoに対するエラーメッセージ文字列を取得するためにstrerrorという関数がありますが，BC++では返される文字列の末尾に改行文字が追加されています．MS-Cでは末尾に改行文字は追加されていません．移植性に配慮する場合は，戻された文字列の末尾が改行であるかどうかを調べて，改行であればヌル文字に置き換えるという処理が必要でしょう．

【clock】

BC++ではCLOCKS_PER_SECが1.0となっており，MS-Cでは1000が定義されています．clock()の戻り値をCLOCKS_PER_SECで割ることでプログラム開始時からの経過秒数を得られることになっていますが，この結果はBC++ではdouble型になりMS-Cではlong型になります．移植性に配慮するならば，除算の結果は(double)でキャストしておくべきでしょう．

【DOSファンクション】

MS-CではDOSファンクションは，ほとんど_dos_xxxという形式で統一されています．BC++では，_dos_という前置きなしに名前がつけられています．MS-Cでは，ほとんどの_dos_関数が戻り値0で正常終了をあらわし，0以外でDOSのエラーコードをあらわします．DOSファンクション自身が戻り値を持つ場合は，それを格納するためのポインタも関数の引数として渡します．BC++では，関数によって適当な戻り値が異なり，DOSのエラーコードは_doserrnoというグローバル変数で判断するようになっています．

全体的に，MS-Cのほうが形式は統一されているかわり使うための手間が増えることがあり，BC++のほうが形式が統一されていないかわり関数自身がすっきりしている，というところでしょうか．実際に，どちらが便利であるかは好みや慣れの問題があるでしょう．ただし，移植性に配慮するならば，両方の関数を用意しなければなりません．

今回は，ほとんどの関数を作成したのでたいていの関数については吸収することができたと思います．ただし，BC++ではabsread，abswriteなど直接ディスクを操作する関数も用意されているのですが，対応するMS-Cの関数が存在しないのとプログラムミスや入力ミスがディスクの破壊につながりかねないのでサポートしていません．また，intr関数もBC++ではINT 25H/26Hのときの特別な処理を行っていますが，今回作成した関数ではその対応を行っていません．こうしたプログラムがほんとうに必要なときは，インラインアセンブラを使うか，アセンブラを使って記述してください．

ファイルを扱う関数については，若干注意が必要です．MS-Cでは，低水準入出力関数とDOSファンクションについて混在して扱うことができますが，BC++では必ずどちらかに統一しなければならないからです．これは，BC++の_openやopen関数がアクセスモードを保持している内部変数が設定されるまでは，低水準入出力関数は正しくこれらのファイルハンドルを扱えないからです．たとえば，_dos_openでオープンしたファイルハンドルからreadでデータを読み込むことはできません．

致命的エラーハンドラを扱う関数もあります．これは，設定するハンドラが通常の関数かfar関数かという違いがあります．

【数学関数】

MS-Cではlong double型に対応した数学関数がサポートされています．すべての関数名には末尾にl（エル）がつけられていて，引数と戻り値としてdoubleのかわりにlong double型を使い，より精度の高い演算ができるようになっています．

BC++で互換ライブラリを作る場合，もし精度まで追及して作成しようとするとエミュレータ自身を改善しなければならなくなります．そこまで手間をかけるならば，BC++ 3.0を使ったほうがはるかに楽です．今回は，単にdouble型に直して演算させるだけにしました．したがって，ソースレベルではコンパイル可能ですが，実際の精度はあくまでもdoubleのレベルに留まります．表現できる値の範囲もdoubleの範囲（約10308くらい）に限られます．また，long double型の関数から呼び出されるエラーハンドラはmatherrlでなくmatherrです．

MS-Cには，j0，j1，jn，y1，y2，ynというベッセル関数もありますが，今回はこれらはサポートしませんでした．まず，使う人はいないでしょうし，

Cプログラミング用語
バイナリモード　ファイルに対して制御文字を含む全ての文字列を記憶，読みだしができる．

なにより計算式を調べるのがめんどうだったのです．

【ディレクトリ操作】

BC++のsetdisk, getdiskは，MS-Cでは_chdrive, _getdriveという名前になっています．ディスクドライブと対応する数値では注意すべき点があります．BC++ではDOSファンクションが使っている形式をそのまま使っているので，関数によって0がAドライブをあらわしたり，0はカレントドライブで1がAドライブをあらわすことがあります．しかし，MS-Cではこうした違いをライブラリが吸収してすべて0がカレントドライブ，1がAドライブをあらわすようになっています．もともとDOSファンクション自身で統一がとれていない点が問題なのですが，setdiskと_chdriveでも，それぞれAドライブをあらわす数値に0と1という違いがあるのです．

【ファイル名操作】

BC++にもMS-Cにもパス名を，ドライブ名/ディレクトリ名/ファイル名/拡張子に分解したり逆に構築するための関数があります．それぞれ，fnsplitと_splitpath，fnmergeと_makepathという名前です．fnsplitと_splitpathは，同じような機能をもっていますが，fnmergeと_makepathでは少し違いがあります．_makepathは，ドライブ名の後ろの':'がなくてもかまいませんが，fnmergeには必要です．拡張子も_makepathでは'.'がなくてもかまいませんが，fnmergeには必要です．

MS-Cでサポートされている_fullpathに相当する関数はBC++にはありません．MS-Cでサポートされている_searchenvは，BC++では環境変数PATHのみを対象にしたsearchpathという関数に相当するでしょう．

なお，これらの関数はBC++ではdir.hで，MS-Cではstdlib.hでプロトタイプが宣言されています．

【メモリ管理】

MS-Cでは，どのメモリモデルでもnearヒープとfarヒープが存在します．メモリモデルによってデフォルトのmallocやreallocがどちらのヒープを利用するかが違いますが，_fmallocや_nmallocはそれぞれ必ずfarヒープとnearヒープにアクセスすることになります．BC++では，スモールデータモデルではnearヒープとfarヒープが存在し，デフォルトではnearヒープが使われますが，ラージデータモデルではfarヒープだけが存在し，つねにヒープはそこから確保されます．64KBを超える領域を確保する場合は，hugeポインタに関してすでに述べたような違いがありますので，まったく同等に扱うことはできません．今回は，64KBを超えるデータが必要な場合に必ず使われるであろうfarmalloc/farcallocとhallocの置き換えだけを行いました．

それぞれがヒープを検査する関数を用意していますが，その形式は異なります．これらはそれぞれのヒープ管理の方法の内部情報を調べなければ互換ライブラリを作成することはできません．今回は，見送ることにしました．

また，MS-Cにはスタックから動的メモリを確保するためのalloca, stackavailがありますがこれも今回は見送りました（BC++ 3.0ではサポートされています）．MS-Cでサポートされている_basedポインタのための関数はまったく考慮していません．

【BIOS関数】

BIOS関数は，両者の間でかなり違いがあります．BC++ではpc98xxxという名前ですが，MS-Cでは_bios_xxxという名前になっています．とくに，BC++の場合はBIOSに限らず積極的にいろいろなファンクションをサポートしています．マウスやファンクションキーの設定，表示の有無，CRT，グラフィックス表示の制御などはMS-CのBIOS関数ではサポートされていません．しかし，ほとんどの場合はBIOSや拡張ファンクションによる呼び出しで実現できるので，互換ライブラリを作るのはそれほど難しくはありません．

ただし，BC++でサポートされている通信ライブラリはこの限りではありません．BIOS呼び出しによるもののほか，独自にすべての制御を行い各種のフロー制御までサポートした高度な関数群がサポートされているためです．これをサポートしようとするとかなりの手間になりますので，今回はサポートしていません．必要ならば，本誌1992年2月号などを参考にして独自に通信関数を作ってみてください．

【テキスト/グラフィックス関数】

PC-9801のテキスト画面やグラフィックス画面を直接操作するような関数は，両者の間でまったく異なります．とくにオリジナルのライブラリが提供する機能を生かそうとするならば，それぞれの処理系用のプログラムを2種類記述しておかなければならないでしょう．今回はテキスト/グラフィックス関数への対応は見送ることにしました．

【ファイルストリームとsetmode】

MS-Cでは，ストリームのアクセス

モードを変更するためにsetmodeを使うことができるようですが，BC++ではこの方法は正しく機能しないことがあります．BC++では，高水準入出力関数と低水準入出力関数で個別にファイルのアクセスモードを管理しているため，ファイルストリームに対しても，直接バイナリかテキストのどちらのモードでアクセスするかを指示しなければなりません．たとえばリスト5のような関数を用意しておくとよいでしょう．

リスト5

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>

int fsetmode(FILE *fp, int mode)
{
    setmode(fileno(fp), mode);
#if defined(__TURBOC__)
    if (mode & O_BINARY)
        stream->flags |= _F_BIN;
    else
        stream->flags &= ~_F_BIN;
#endif
}
```

【その他】

MS-Cには，vscanf，vsscanf，vfscanfがありません．MS-Cの書式解析ルーチンはあまり汎用的に作成されていないので，今回はサポートしていません．

MS-Cでは，sys¥stat.hを使うために必ずsys¥types.hをインクルードしなければなりませんが，BC++ではsys¥stat.hだけで使うことができます．移植性に配慮する場合は，必ず両方をインクルードするようにしてください．

MS-Cにあるsys¥utime.hやsys¥locking.hはBC++にはありません．今回はサポートしていません．

Learning C　互換ライブラリの作成

前節で述べた内容を元に，今回はBC++でもMS-Cでもコンパイルできるよう両者がサポートしているライブラリをできるだけ取り入れました．もっとも，BC++はバージョン3.0でほとんどのMS-Cの関数を取り入れていますので，BC++ 3.0をお使いの方にはあまり必要はないかもしれません．BC++ 3.0では，定義済みマクロ__TURBOC__が0x400と定義されているので，それに合わせて不要な部分は取り込まないように配慮しています．

サポートされている関数，されていない関数についてはすでに述べたとお

リスト6　dual.h

```
/*
 *  dual.h:
 *      Microsoft C と Borland C++ のマクロ、構造体、関数の定義を
 *      共通化するためのヘッダファイル。
 *
 *  Copyright (C) 1992 Motohisa Ohno. All rights reserved.
 */
#if !defined(__DUAL_H)
#define __DUAL_H

#if defined(__TURBOC__)

/*
 *  Borland C++ のための定義
 */

/* 修飾子 */
#define fortran pascal
#define _fortran _pascal
#if (!defined(__BORLANDC__) || (__BORLANDC__ < 0x400))
#define _fastcall _pascal
#endif

/* 割り込みハンドラの引数 */
#define INTR_ARGS unsigned bp_, unsigned di_, unsigned si_, unsigned ds_, ¥
    unsigned es_, unsigned dx_, unsigned cx_, unsigned bx_, ¥
    unsigned ax_, unsigned ip_, unsigned cs_, unsigned flags_

#if (__TURBOC__ < 0x400)

/* ファイル操作 */
#if defined(__STDIO_H)
int rmtmp(void);

/* _fsopen, tempnam not supported */
#endif

/* 数値計算・変換 */
#if defined(__STDLIB_H)
#define _strtold(__str, __endp)  ((long double)strtod(__str, __endp))
#endif
#if defined(__MATH_H) || defined(__STDLIB_H)
long double _atold(const char *__str);
#endif

#if defined(__MATH_H)
struct _complexl {
    long double x, y;
};
int dmsbintoieee(double *src, double *dst);
int dieeetomsbin(double *src, double *dst);
int fmsbintoieee(float *src, float *dst);
int fieeetomsbin(float *src, float *dst);
#define acosl(__x)  ((long double)acos((double)__x))
#define asinl(__x)  ((long double)asin((double)__x))
#define atanl(__x)  ((long double)atan((double)__x))
#define atan2l(__x, __y) ((long double)atan2((double)__x, (double)__y))
#define cabsl(__z)  ((long double)hypot((double)(__z).x, (double)(__z).y))
#define ceill(__x)  ((long double)ceil((double)__x))
#define cosl(__x)   ((long double)cos((double)__x))
#define coshl(__x)  ((long double)cosh((double)__x))
#define expl(__x)   ((long double)exp((double)__x))
#define fabsl(__x)  ((long double)fabs((double)__x))
#define floorl(__x) ((long double)floor((double)__x))
#define fmodl(__x, __y)   ((long double)fmod((double)__x, (double)__y))
#define frexpl(__x, __np) ((long double)frexp((double)__x, __np))
#define hypotl(__x, __y)  ((long double)hypot((double)__x, (double)__y))
#define ldexpl(__x, __n)  ((long double)ldexp((double)__x, __n))
#define logl(__x)   ((long double)log((double)__x))
#define log10l(__x) ((long double)log10((double)__x))
long double modfl(long double __x, long double *__yptr);
#define powl(__x, __y)  ((long double)pow((double)__x, (double)__y))
#define sinl(__x)   ((long double)sin((double)__x))
#define sinhl(__x)  ((long double)sinh((double)__x))
#define sqrtl(__x)  ((long double)sqrt((double)__x))
#define tanl(__x)   ((long double)tan((double)__x))
#define tanhl(__x)  ((long double)tanh((double)__x))

```

Cプログラミング用語
バグ　プログラム中の誤動作箇所を差す．米国ではバグから「テントウ虫」を連想し，デバッガのアイコンなどに使われている．

```
/* j0, j1, jn, y0, y1, yn not supported */
#endif /* defined(__MATH_H) */

/* ディレクトリ操作 */
#if defined(__DIR_H)

#define _chdrive(__drv)  setdisk(__drv - 1)
char *_getdcwd(int __drv, char *__buf, int __len);
#if !defined(_getdrive)
#define _getdrive()  (getdisk() + 1)
#endif
#endif /* defined(__DIR_H) */

/* ファイル名操作 */

#if defined(__STDLIB_H)
#include <dir.h>
#include <dos.h>
#if !defined(_MAX_PATH)
#define _MAX_PATH   MAXPATH
#define _MAX_DRIVE  MAXDRIVE
#define _MAX_DIR    MAXDIR
#define _MAX_FNAME  MAXFNAME
#define _MAX_EXT    MAXEXT
#endif

char *_fullpath(char *buffer, const char *path, size_t len);
#if !defined(_splitpath)
#define _splitpath(__path, __drv, __dir, __file, __ext) ¥
     (void)fnsplit(__path, __drv, __dir, __file, __ext)
#endif
void _makepath(char *__path, const char *__drv, const char *__dir,
          const char *__file, const char *__ext);
void _searchenv(const char *name, const char *env, char *path);
#endif /* defined(__STDLIB_H) */

#if defined(__ALLOC_H)
/* alloca, stackavail, _expand, _bxxx, _nxxx, etc. not supported */
#endif /* defined(__ALLOC_H) */

#if defined(__BIOS_H)

#define _KEYBRD_READ        0
#define _KEYBRD_READY       1
#define _KEYBRD_SHIFTSTATUS 2
#define _KEYBRD_INITIALIZE  3
#define _KEYBRD_SENSE       4

#define _PRN_INIT           0x10
#define _PRN_WRITE          0x11
#define _PRN_STRING         0x30
#define _PRN_STATUS         0x12

#define _TIME_GETCLOCK      0x00
#define _TIME_SETCLOCK      0x01

#define _bios_memsize() pc98mem()
unsigned _bios_equiplist(void);
unsigned _bios_keybrd(unsigned __cmd, unsigned char *__ktbl);
unsigned _bios_printer(unsigned __cmd, unsigned char *__data);
unsigned _bios_timeofday(unsigned, char *);
#endif

#if defined(__DOS_H)

#if !defined(_HARDERR_IGNORE)
#define _HARDERR_IGNORE     0
#define _HARDERR_RETRY      1
#define _HARDERR_ABORT      2
#define _HARDERR_FAIL       3
#endif /* !defined(_HARDERR_IGNORE) */

#if !defined(__MSC)
#define _A_NORMAL   FA_NORMAL
#define _A_RDONLY   FA_RDONLY
#define _A_HIDDEN   FA_HIDDEN
#define _A_SYSTEM   FA_SYSTEM
#define _A_VOLID    FA_LABEL
#define _A_SUBDIR   FA_DIREC
#define _A_ARCH     FA_ARCH

struct find_t {
    char    reserved[21];
    char    attrib;
    unsigned wr_time;
    unsigned wr_date;
    long     size;
    char     name[13];
};

#define _dos_findfirst(__path, __attr, __brk) ¥
   (unsigned)findfirst(__path, (struct ffblk *)__brk, __attr)
#define _dos_findnext(__brk) (unsigned)findnext((struct ffblk *)__brk)
#define _disable() disable()
#define _enable() enable()
#endif /* !defined(__MSC) */

struct dostime_t {
    unsigned char hour;
    unsigned char minute;
    unsigned char second;
    unsigned char hsecond;
};

struct dosdate_t {
    unsigned char day;
    unsigned char month;
    unsigned int  year;
    unsigned char dayofweek;
};

struct diskfree_t {
    unsigned total_clusters;
    unsigned avail_clusters;
    unsigned sectors_per_cluster;
    unsigned bytes_per_sector;
};

void _chain_intr(void (interrupt far *__handler)());
unsigned _dos_allocmem(unsigned size, unsigned *segp);
unsigned _dos_close(int handle);
unsigned _dos_creat(const char *path, unsigned attr, int *handle);
unsigned _dos_creatnew(const char *path, unsigned attr, int *handle);
unsigned _dos_freemem(unsigned segp);
void _dos_getdate(struct dosdate_t *datep);
void _dos_getdrive(unsigned *drive);
unsigned _dos_getdiskfree(unsigned drive, struct diskfree_t *dinfo);
unsigned _dos_getfileattr(const char *file, unsigned *attr);
unsigned _dos_getftime(int handle, unsigned *datep, unsigned *timep);
void _dos_gettime(struct dostime_t *timep);
#define _dos_getvect(__inum) getvect((int)__inum)
#define _dos_keep(__status, __size)  keep(__status, __size)
unsigned _dos_open(const char *path, unsigned flags, int *handle);
unsigned _dos_read(int handle, void _far *buf, unsigned len, unsigned *nread);
unsigned _dos_setblock(unsigned size, unsigned segx, unsigned *maxp);
unsigned _dos_setdate(struct dosdate_t *datep);
void _dos_setdrive(unsigned drive, unsigned *ndrives);
unsigned _dos_setfileattr(const char *file, unsigned attr);
unsigned _dos_setftime(int handle, unsigned date, unsigned time);
unsigned _dos_settime(struct dostime_t *timep);
#define _dos_setvect(inum, isr)  setvect((int)inum, isr)
unsigned _dos_write(int handle, const void _far *buf, unsigned len,
          unsigned *nwrite);.
void _harderr(void (far *__handler)());
#define _hardresume(__ax) hardresume(__ax)
#define _hardretn(__ret)  hardretn(__ret)
#endif /* defined(__DOS_H) */

#if defined(__TIME_H)
char *_strdate(char *__str);
char *_strtime(char *__str);
#endif

#endif /* __TURBOC__ < 0x400 */


#else /* !defined(__TURBOC__) */

/*
 *  Microsoft C のための定義
 */

/* 識別子 */
#if defined(M_I86TM)
#define __SMALL__
#endif
#if defined(M_I86SM)
#define __SMALL__
#endif
#if defined(M_I86MM)
#define __MEDIUM__
#endif
#if defined(M_I86CM)
#define __COMPACT__
#endif
#if defined(M_I86LM) && !defined(M_I86HM)
#define __LARGE__
#endif
#if defined(M_I86HM)
#define __HUGE__
#endif

/* 割り込みハンドラの引数 */
#define INTR_ARGS unsigned es_, unsigned ds_, unsigned di_, unsigned si_, ¥
     unsigned bp_, unsigned sp_, unsigned bx_, unsigned dx_, unsigned cx_, ¥
     unsigned ax_, unsigned ip_, unsigned cs_, unsigned flags_

#if defined(_REGS_DEFINED)
/* dos.h */
#include <stdio.h>

#if !defined(_REGPACK_DEFINED)
#define _REGPACK_DEFINED
/*
 *  intr 関数で使われるパックレジスタ構造体
 */
struct REGPACK {
    unsigned r_ax, r_bx, r_cx, r_dx;
    unsigned r_bp, r_si, r_di, r_ds, r_es, r_flags;
};
#endif /* !defined(_REGPACK_DEFINED) */

#if !defined(_FFBLK_DEFINED)
#define _FFBLK_DEFINED
/*
 *  ワイルドカード検索ルーチン
 */
struct ffblk {
    char ff_reserved[21];
    char ff_attrib;
    unsigned ff_ftime;
    unsigned ff_fdate;
    long ff_fsize;
    char ff_name[13];
};
#endif /* !defined(_FFBLK_DEFINED) */

#if !defined(_DFREE_DEFINED)
#define _DFREE_DEFINED
/*
 *  getdfree で使われる構造体
 */
struct dfree {
    unsigned df_avail;
    unsigned df_total;
    unsigned df_bsec;
```

C プログラミング用語

ヒープ　ヒープは伸び縮みする動的データ領域でメモリ上にはスタック領域の後ろに確保される。

```
    unsigned df_sclus;
};
#endif /* !defined(_DFREE_DEFINED) */

#if !defined(_FATINFO_DEFINED)
#define _FATINFO_DEFINED
/*
 *  getfat, getdfat で使われる構造体
 */
struct fatinfo {
    char    fi_sclus;
    char    fi_fatid;
    int     fi_nclus;
    int     fi_bysec;
};
#endif /* !defined(_FATINFO_DEFINED) */

#if !defined(_DATE_DEFINED)
#define _DATE_DEFINED
/*
 *  getdate などで使われる日付をあらわす構造体
 */
struct date {
    int  da_year;
    char da_day;
    char da_mon;
};
#endif /* !defined(_DATE_DEFINED) */

#if !defined(_TIME_DEFINED)
#define _TIME_DEFINED
/*
 *  gettime などで使われる時刻をあらわす構造体
 */
struct time {
    unsigned char ti_min;
    unsigned char ti_hour;
    unsigned char ti_hund;
    unsigned char ti_sec;
};
#endif /* !defined(_TIME_DEFINED) */

/*
 *  疑似レジスタ変数をまねるための構造体定義
 */
struct __tag_pseudo_regs {
    union {
        unsigned char b[2];
        unsigned w;
    } reg_ax;
    union {
        unsigned char b[2];
        unsigned w;
    } reg_bx;
    union {
        unsigned char b[2];
        unsigned w;
    } reg_cx;
    union {
        unsigned char b[2];
        unsigned w;
    } reg_dx;
    unsigned reg_si;
    unsigned reg_di;
    unsigned reg_bp;
    unsigned reg_ds;
    unsigned reg_es;
    unsigned reg_flags;
};
extern struct __tag_pseudo_regs __pseudo_regs;

#define _AX (__pseudo_regs.reg_ax.w)
#define _AH (__pseudo_regs.reg_ax.b[1])
#define _AL (__pseudo_regs.reg_ax.b[0])
#define _BX (__pseudo_regs.reg_bx.w)
#define _BH (__pseudo_regs.reg_bx.b[1])
#define _BL (__pseudo_regs.reg_bx.b[0])
#define _CX (__pseudo_regs.reg_cx.w)
#define _CH (__pseudo_regs.reg_cx.b[1])
#define _CL (__pseudo_regs.reg_cx.b[0])
#define _DX (__pseudo_regs.reg_dx.w)
#define _DH (__pseudo_regs.reg_dx.b[1])
#define _DL (__pseudo_regs.reg_dx.b[0])
#define _SI (__pseudo_regs.reg_si)
#define _DI (__pseudo_regs.reg_di)
#define _BP (__pseudo_regs.reg_bp)
#define _DS (__pseudo_regs.reg_ds)
#define _ES (__pseudo_regs.reg_es)
#define _FLAGS (__pseudo_regs.reg_flags)
void geninterrupt(int __intno);

/*
 *  ｆａｒポインタ用のマクロ
 */
#if defined(FP_SEG)
#undef FP_SEG
#undef FP_OFF
#define FP_SEG(__fp) ((unsigned)((unsigned long)(__fp) >> 16))
#define FP_OFF(__fp) ((unsigned)(unsigned long)(__fp))
#define MK_FP(__seg, __ofs) ¥
    (((unsigned long)(unsigned)(__seg)<<16) | (unsigned)(__ofs))
#endif /* defined(FP_SEG) */

/*
 *  汎用関数
 */
int allocmem(unsigned __size, unsigned *__segp);
int bdosptr(int __dosfun, void *__ptr, unsigned __dosal);
void ctrlbrk(int (*handler)(void));
void delay(unsigned __milliseconds);
#define disable() _disable()
#define enable() _enable()
#define freemem(__segx) (unsigned)freemem(__segx)
#define getcbrk() (bdos(0x33, 0, 0))
void getdate(struct date *__datep);
void getdfree(unsigned char __drive, struct dfree * dtable);
void getfat(unsigned char __drive, struct fatinfo *__dtable);
#define getfatd(__dtable) getfat(0, __dtable)
unsigned getpsp(void);
void setswitchar(char ch);
void gettime(struct time *__timep);
int getverify(void);
void harderr(int (*handler)());
#define hardresume(__ax)  _hardresume(__ax)
#define hardretn(__ret)   _hardretn(__ret)
void intr(int intno, struct REGPACK *regs);
#define keep(__status, __size)  _dos_keep(__status, __size)
void nosound(void);
int setblock(unsigned __segx, unsigned __newsize);
#define setcbrk(cbrk) (bdos(0x33, (cbrk), 1))
void setdate(struct date *__datep);
void setswitchar(char __ch);
void settime(struct time *__timep);
void setverify(int __value);
void sleep(unsigned __seconds);
void sound(unsigned __frequency);
char far *getdta(void);
void setdta(char far *__dta);
#define getvect(__inum) _dos_getvect((int)__inum)
#define setvect(inum, isr)  _dos_setvect((int)inum, isr)
#define peek(__s,__o)   (*((int  far *)MK_FP((__s),(__o))))
#define peekb(__s,__o)  (*((char far *)MK_FP((__s),(__o))))
#define poke(__s, __o, __d)  (*((int far *)MK_FP((__s),(__o)))=(int)(__d))
#define pokeb(__s, __o, __d) (*((char far *)MK_FP((__s),(__o)))=(char)(__d) )

#if !defined(findfirst)
#define findfirst(path,ffblk,attrib) ¥
    (-(int)_dos_findfirst(path,attrib,(struct find_t *)ffblk))
#define findnext(ffblk) (-(int)_dos_findnext((struct find_t *)ffblk))
#endif

int inp(unsigned __port);
unsigned inpw(unsigned __port);
int outp(unsigned __port, int __data);
unsigned outpw(unsigned __port, unsigned __data);

#define inportb(__port)  inp(__port)
#define inport(__port)   inpw(__port)
#define outportb(__port, __data)  outp(__port, __data)
#define outportw(__port, __data)  outpw(__port, __data)

#endif /* defined(_REGS_DEFINED) */

#if !defined(_FTIME_DEFINED)
#define _FTIME_DEFINED
/*
 *  ファイルのタイムスタンプをあらわす構造体
 */
struct ftime {
    unsigned ft_tsec  : 5;   /* Two second interval */
    unsigned ft_min   : 6;   /* Minutes */
    unsigned ft_hour  : 5;   /* Hours */
    unsigned ft_day   : 5;   /* Days */
    unsigned ft_month : 4;   /* Months */
    unsigned ft_year  : 7;   /* Year */
};
#endif /* !defined(_FTIME_DEFINED) */

int _chmod(const char *__path, int __func, ...);
int _close(int __handle);
int _creat(const char *__path, int __attr);
int creatnew(const char *__path, int __mode);
int creattemp(char *__path, int __amode);
int getftime(int __handle, struct ftime *__ftimep);
int ioctl(int __handle, int __func, ...);
int _open(const char *__path, int __oflags);
int _read(int __handle, void *__buf, unsigned __len);
int setftime(int __handle, struct ftime *__ftimep);
int _write(int __handle, void *__buf, unsigned __len);

#if defined(EXIT_FAILURE)
/* stdlib.h */
#include <time.h>
#define random(num)  (int)(((long)rand()*(num))/((unsigned)RAND_MAX+1))
#define randomize()  srand((unsigned)time(NULL))
#endif

#if defined(_EXCEPTION_DEFINED)
/* 数学関数 */
double poly(double __x, int __degree, double __coeffs[]);
#define pow10(__num) pow(10.0, (__num))
#endif

#endif  /* defined(__TURBOC__) */

#endif  /* __DUAL_H */
```


C プログラミング用語

引数　関数や手続きの呼び出し時，引数で値を渡す．

りです．かなり，多くの内容を取り込んでしまったためファイルも大きなものになってしまいました．もちろん関数ごとに分割してライブラリを作成するほうが効率はずっとよくなります．

基本的には，ほとんどの定義がdual.hに，実際の関数がdual.cに入っています．MS-CになくBC++だけに存在するヘッダファイルも用意しました．MS-Cユーザーの方は，次のヘッダファイルをMS-Cの標準ヘッダファイルのあるディレクトリにコピーしておいてください．

alloc.h
dir.h
mem.h
pc98.h

dual.hは，作成するプログラム中のすべての# include文の後ろに記述してください．順番が入れ換わると，必要な定義が抜けてしまうことがあります．

また，BC++でMS-Cスタイルのコンソール入出力をサポートするためにdual.cとは別にconio.cを用意しました．DOSの低レベルファンクションだけを使ってコンソールとのやり取りを行う場合は，このモジュールをリンクしてください．conio.cを使うことでMS-Cスタイルに置き換えられるコンソール入出力関数は，次のとおりです．

cgets
cprintf
cputs
cscanf
getch
getche
kbhit
putch
ungetch

MS-Cで困ったのは，ヘッダファイルがインクルードされているかどうか判別する方法が用意されていないことでした．メモリとコンパイル時間の節約のため，ヘッダファイルがインクルードされていなければそれに関係ない定義も除きたいためです．MS-Cでは，構造体の重複定義を防ぐためそれぞれに識別子を用意していますから使えるものはそれを使いましたが，io.hなどはまったく判別方法がないので関連する関数のプロトタイプ宣言は常に有効になっています．

dual.hでは，互換ライブラリという以外にも若干よけいな定義が行われています．マクロINTR_ARGSはinterruptで修飾された割り込みハンドラの引数リストを定義しています．これを使うことで，どちらの処理系でもリスト13の記述によって正しくレジスタにアクセスできるようになりま

リスト7．dual.c

```
/*
 *  dual.c:
 *      移植性のための補助ルーチン
 *
 *  Copyright (C) 1992 Motohisa Ohno, All rights reserved.
 */

#if defined(__TURBOC__)
/*
 *  Borland C++ のための互換ルーチン
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <io.h>
#include <fcntl.h>
#include <dir.h>
#include <dos.h>
#include <bios.h>
#include <pc98.h>
#include "dual.h"

/*
 *  すべてのテンポラリファイルをクローズ／削除する
 */
int rmtmp(void)
{
    int i, num;
    FILE *fp;

    /* ストリームのテーブルを持つ非公開変数を調べる */
    for (i = num = 0, fp = _streams; i < FOPEN_MAX; i++, fp++) {
        if (fp->fd >= 0 && fp->istemp != 0) {
            if (fclose(fp) == 0)
                num++;
        }
    }
    return num;
}

/*
 *  IEEE 形式の倍精度実数
 */
struct ieee_double {
    unsigned mag6:4;    /* 仮数部５２ビット */
    unsigned mag5:8;
    unsigned mag4:8;
    unsigned mag3:8;
    unsigned mag2:8;
    unsigned mag1:8;
    unsigned mag0:8;
    unsigned exp:11;    /* 指数部１１ビット */
    unsigned sign:1;    /* 符号１ビット */
};

/*
 *  Microsoft 形式の倍精度実数
 */
struct msbin_double {
    unsigned mag7:3;    /* 仮数部５５ビット */
    unsigned mag6:4;
    unsigned mag5:8;
    unsigned mag4:8;
    unsigned mag3:8;
    unsigned mag2:8;
    unsigned mag1:8;
    unsigned mag0:8;
    unsigned sign:1;    /* 符号１ビット */
    unsigned exp:8;     /* 指数部８ビット */
};

/*
 *  IEEE 形式の単精度実数
 */
struct ieee_float {
    unsigned mag2:7;    /* 仮数部２３ビット */
    unsigned mag1:8;
    unsigned mag0:8;
    unsigned exp:8;     /* 指数部８ビット */
    unsigned sign:1;    /* 符号１ビット */
};

/*
 *  Microsoft 形式の単精度実数
 */
```

Cプログラミング用語
標準エラー出力　標準エラー出力ストリームの結合先．デフォルトではディスプレイが指定されている．

す．

さらに，MS-CでBC++のレジスタ疑似変数をまねるために対応する名前のマクロを定義し，geninterrupt関数でその内容を使ったソフトウェア割り込みを実行するようにしています．しかし，BC++のように直接レジスタにアクセスできるわけではありませんから効率があがるわけではありません．新たにプログラムを記述する場合はintrなどを使ったほうがよいでしょう．

なお，作成したライブラリ関数はそれぞれの処理系のものを必ずしも同じレベルで再現したものばかりではありません．いくつかの関数は，実際のCのプログラムでなくヘッダファイル中のマクロで定義することですませているものがあります．いくつかの関数は，プログラムの簡略化のため仕様に制限を設けているものがあります．また，とくに戻り値については必ずしもお互いのライブラリと同じ内容を反映するとは限りませんので注意してお使いください．

各関数の詳細については，各ソースコードのコメントを参考にしてください．

リスト8．alloc.h

```
/*
 *  alloc.h - Borland C++ 互換のためのヘッダファイル
 */

#if     !defined(__ALLOC_H)
#define __ALLOC_H

#include <malloc.h>
#include <stddef.h>

#if (defined(M_I86SM) || defined(M_I86MM))
/* small data model */
#define coreleft() _memmax()
// farcoreleft not supported
#else
/* large data model */
// coreleft, farcoreleft not supported
#endif

// int brk(void *__addr);       /* not supported */
// void *sbrk(int __incr);      /* not supported */

#define heapcheck()  _heapchk()
#define farheapcheck()  _fheapchk()

#define farcalloc(__num, __siz) ¥
    ((void far *)_halloc((size_t)__num, (size_t)__siz))
#define farfree(__ptr) _ffree(__ptr)
#define farmalloc(__siz) ((void far *)_halloc(__siz, 1))
// void far *farrealloc(void far *ptr, unsigned long siz); /* not supported */

#endif  /* __ALLOC_H */
```

リスト9．dir.h

```
/*
 *  dir.h - Borland C++ 互換のためのヘッダファイル
 */

#if !defined( __DIR_H )
#define __DIR_H

#include <direct.h>
#if !defined(_MAX_PATH)
#include <stdlib.h>
#endif


#if !defined(_FFBLK_DEFINED)
#define _FFBLK_DEFINED
/*
 *  ワイルドカード検索ルーチン
 */
struct ffblk {
    char ff_reserved[21];
    char ff_attrib;
    unsigned ff_ftime;
    unsigned ff_fdate;
    long ff_fsize;
    char ff_name[13];
};
#endif

#define findfirst(__path, __ffblk, __attrib) ¥
    (-(int)_dos_findfirst(__path, __attrib, (struct find_t *)__ffblk))
#define findnext(__ffblk) ¥
    (-(int)_dos_findnext((struct find_t *)__ffblk))

/*
 *  ファイル名操作ルーチン
 */
#define MAXPATH  80
#define MAXDRIVE 3
#define MAXDIR   66
#define MAXFILE  9
#define MAXEXT   5
#define fnsplit(__path, __drv, __dir, __file, __ext) ¥
    _splitpath(__path, __drv, __dir, __file, __ext)
#define fnmerge(__path, __drv, __dir, __file, __ext) ¥
    _makepath(__path, __drv, __dir, __file, __ext)

int getcurdir(int __drive, char *__dir);
#define getdisk() (_getdrive() - 1)
char *searchpath(const char *__file);
#define setdisk(__drv) _chdrive(__drv + 1)

#endif /* !defined(__DIR_H) */
```

リスト10．mem.h

```
/*
 *  mem.h - Borland C++ 互換のためのヘッダファイル
 */
#if !defined( __MEM_H )
#define __MEM_H

#include <memory.h>

#endif  /* !defined(__MEM_H) */
```


Cプログラミング用語
標準入出力　標準入出力ストリームの結合先．デフォルトではディスプレイとキーボードが指定されている．

```c
struct msbin_float {
    unsigned mag2:7;    /* 仮数部2 3 ビット */
    unsigned mag1:8;
    unsigned mag0:8;
    unsigned sign:1;    /* 符号1ビット */
    unsigned exp:8;     /* 指数部8ビット */
};

/*
 *  IEEE 形式の倍精度実数を Microsoft 形式に変換
 */
int dieeetomsbin(double *src, double *dst)
{
    union {
        struct ieee_double i;
        double d;
    } a;
    union {
        struct msbin_double m;
        double d;
    } b;
    int exp;

    a.d = *src;
    b.m.sign = a.i.sign;
    exp = a.i.exp - 1023 + 129;
    if (exp > 255) return 1;
    if (exp < 0) {
        *dst = 0.0;
        return 0;
    }
    b.m.exp = exp;
    b.m.mag0 = a.i.mag0;
    b.m.mag1 = a.i.mag1;
    b.m.mag2 = a.i.mag2;
    b.m.mag3 = a.i.mag3;
    b.m.mag4 = a.i.mag4;
    b.m.mag5 = a.i.mag5;
    b.m.mag6 = a.i.mag6;
    b.m.mag7 = 0;
    *dst = b.d;
    return 0;
}

/*
 *  Microsoft 形式の倍精度実数を IEEE 形式に変換
 */
int dmsbintoieee(double *src, double *dst)
{
    union {
        struct ieee_double i;
        double d;
    } a;
    union {
        struct msbin_double m;
        double d;
    } b;

    b.d = *src;
    a.i.sign = b.m.sign;
    a.i.exp = b.m.exp - 129 + 1023;
    a.i.mag0 = b.m.mag0;
    a.i.mag1 = b.m.mag1;
    a.i.mag2 = b.m.mag2;
    a.i.mag3 = b.m.mag3;
    a.i.mag4 = b.m.mag4;
    a.i.mag5 = b.m.mag5;
    a.i.mag6 = b.m.mag6;
    *dst = a.d;
    return 0;
}

/*
 *  IEEE 形式の単精度実数を Microsoft 形式に変換
 */
int fieeetomsbin(float *src, float *dst)
{
    union {
        struct ieee_float i;
        float f;
    } a;
    union {
        struct msbin_float m;
        float f;
    } b;
    int exp;

    a.f = *src;
    b.m.sign = a.i.sign;
    exp = a.i.exp - 127 + 129;
    if (exp > 255) return 1;
    b.m.exp = exp;
    b.m.mag0 = a.i.mag0;
    b.m.mag1 = a.i.mag1;
    b.m.mag2 = a.i.mag2;
    *dst = b.f;
    return 0;
}

/*
 *  Microsoft 形式の単精度実数を IEEE 形式に変換
 */
int fmsbintoieee(float *src, float *dst)
{
    union {
        struct ieee_float i;
        float f;
    } a;
    union {
        struct msbin_float m;
        float f;
    } b;
    int exp;

    b.f = *src;
    a.i.sign = b.m.sign;
    exp =  b.m.exp - 129 + 127;
    if (exp < 0) {
        *dst = 0.0F;
        return 1;
    }
    a.i.exp = exp;
    a.i.mag0 = b.m.mag0;
    a.i.mag1 = b.m.mag1;
    a.i.mag2 = b.m.mag2;
    *dst = a.f;
    return 0;
}

/*
 *  文字列を long double 値に変換する
 */
long double _atold(const char *str)
{
    long double a;

    sscanf(str, "%Lf", &a);
    return a;
}

/*
 *  long double 対応の商／剰余の計算
 */
long double modfl(long double x, long double *yptr)
{
    double a, b;

    a = modf((double)x, &b);
    *yptr = (long double)b;
    return (long double)a;
}

/*
 *  指定したドライブのカレントディレクトリを取得
 *      drive: ドライブ番号 (1=A:, 2=B:, 3=C:, ...)
 *      buf: ディレクトリを保存するバッファ
 *      len: バッファの長さ
 *  戻り値:
 *      NULL以外: 取得したディレクトリ名へのポインタ
 *      NULL: バッファの長さが不足している
 */
char *_getdcwd(int drive, char *buf, int len)
{
    char dbuf[MAXPATH];

    if (drive == 0)
        drive = getdisk() + 1;
    dbuf[0] = 'A' + drive - 1;
    dbuf[1] = ':';
    dbuf[2] = '¥¥';
    if (getcurdir(drive, dbuf + 3) < 0)
        return NULL;
    if (strlen(dbuf) >= len) return NULL;
    return buf ? strcpy(buf, dbuf) : strdup(buf);
}

/*
 *  指定したファイル名のフルパス名を取得
 *      buf: 結果を返すためのバッファ
 *      path: ファイル名
 *      len: buf が示すバッファの長さ
 *  戻り値
 *      NULL以外: 得られたパス名へのポインタ
 *      NULL: バッファの長さが不足している
 *  ※注意
 *      .¥ や ..¥ などには対応していない
 */
char *_fullpath(char *buf, const char *path, size_t len)
{
    char drv[MAXDRIVE], dir[MAXDIR], file[MAXFILE], ext[MAXEXT];
    char cdrv[MAXDRIVE], cdir[MAXDIR];
    char tmp[MAXPATH];
    int drive, n;

    fnsplit(path, drv, dir, file, ext);
    if (drv[0])
        drive = toupper((unsigned char)drv[0]) - 'A' + 1;
    else {
        drive = 0;
        drv[0] = 'A' + getdisk();
        drv[1] = ':';
        drv[2] = 0;
    }
    if (_getdcwd(drive, tmp, MAXPATH) == NULL)
        return NULL;
    n = strlen(tmp);
    if (n && tmp[n-1] != '¥¥')
        strcat(tmp, "¥¥");
    fnsplit(tmp, cdrv, cdir, NULL, NULL);
    if (drv[0] == 0)
        strcpy(drv, cdrv);
    if (dir[0] == 0)
        strcpy(dir, cdir);
    fnmerge(tmp, drv, dir, file, ext);
    if (strlen(tmp) >= len)
        return NULL;
    return buf ? strcpy(buf, tmp) : strdup(buf);
}

/*
 *  パス名の連結
 *      path: 結果のパス名を格納するバッファ
 *      drv: ドライブ名
 *      dir: ディレクトリ名
 *      file: ファイル名
 *      ext: 拡張子
```

C プログラミング用語
標準補助入出力　標準補助入出力ストリームの結合先。デフォルトでは RS232Cポートが指定されている。

最後に

よく「Cの移植性がよいというのは幻にすぎない」といわれます．それはまったく嘘というわけではありませんが，逆にそこまでいい切る必要もないと思います．実際，今回作成したライブラリを使ってかなり移植性が吸収できる面もあるでしょう．もともと，Cの移植性はUNIXというOSを記述する際にアセンブラと比較して述べられてきたものです．過度な期待は禁物ですが，注意さえしていれば移植性のあるソースコードを記述するのはそんなにたいへんな作業ではないはずです．

言語仕様がメーカーごとに異なった形で拡張されていくのはある意味では残念ですが，そうした競争によってより使いやすいものが手に入るならばそれはそれでよいことだと思います．BC++のようにほかメーカーのものも吸収して取り込んでいこうという動きもありますし，より多くの処理系でプログラムをコンパイルできるようにしておけば，そのソースコードの価値が何倍にもなると思います．そう考えれば，移植性のために少しばかりの手間をかけるのも惜しくはないでしょう．

リスト11．pc98.h

```
/*
 *  pc98.h - Borland C++ 互換のためのヘッダファイル
 */
#if !defined(__PC98_H)
#define __PC98_H

#include <bios.h>
#include <dos.h>
#include "dual.h"

struct mouseinfo {
    unsigned short  reg_ax;
    unsigned short  reg_bx;
    unsigned short  reg_cx;
    unsigned short  reg_dx;
    void *shape;
    void (*fptr)(unsigned __ax, unsigned __bx, unsigned __cx, unsigned __dx);
};

int pc98crt(int __cmd, ...);
unsigned pc98key(int __cmd, ...);
#define pc98mem() _bios_memsize()
int pc98mouse(int __cmd, struct mouseinfo *__mp);
int pc98print(int __cmd, ...);
void (*pc98timer(unsigned interval, void (*fptr)()))();
void pc98getfont(unsigned __code, void *__buffer);
void pc98putuserfont(unsigned __code, void *__buffer);
void pc98fkeyon(void);
void pc98fkeyoff(void);
int pc98fkeychk(void);
void pc98getfkey(int __id, void *__buffer);
void pc98setfkey(int __id, const void *__buffer);

#endif  /* !defined(__PC98_H) */
```

リスト12．conio.c

```
/*
 *  conio.c - Microsoft C 互換のコンソール入出力を実現するためのプログラム
 */

#include <stdio.h>
#include <stdarg.h>
#include <dos.h>
#include <conio.h>

/*
 *  文字列入力
 *      buf[0]: buf + 2 以降のバッファサイズ (sizeof buf - 2)
 *      buf[1]: 実際に入力された文字数が返される (改行を除く)
 *      buf[2]〜: 入力された文字列
 */
char *cgets(char *buf)
{
    bdosptr(0x0A, buf, 0);
    buf[2 + (unsigned char)buf[1]] = 0;
    return buf + 2;
}

/*
 *  書式つき出力
 *      format: 書式
 *      ※出力桁数は 200 文字を超えられない。
 *  戻り値
 *      出力したバイト数
 */
int cprintf(const char *format, ...)
{
    char buf[200];
    int i, n;
    va_list ap;

    va_start(ap, format);
    n = vsprintf(buf, format, ap);
    va_end(ap);
    for (i = 0; i < n; i++)
        putch(buf[i]);
    return n;
}

/*
 *  文字列出力
 *      str: 文字列
 *  戻り値
 *      0
 */
int cputs(const char *str)
{
    while (*str)
        putch(*str++);
    return 0;
}

/*
 *  文字出力
 *      c: 文字コード
 *      ※Microsoft C の仕様にあわせてあるため、文字コード 0xFF は
 *        出力できない。MS-DOS の入力バッファに文字が残っているときに、
 *        文字コード 0xFF を出力すると、入力が1文字ぶんスキップされる。
 *  戻り値
 *      表示した文字コード
 */
int putch(int c)
{
    bdos(6, c, 0);
    return c;
}
```

リスト13．

```
void interrupt far handler(INTR_ARGS) { ... }
```

C プログラミング用語

プリプロセッサ　ソースファイルの一部分をある条件で加工したり，読み飛ばしたり，他のソースファイルを読み込むプログラム．

```
 */
void _makepath(char *path, const char *drv, const char *dir,
        const char *file, const char *ext)
{
    char _drv[MAXDRIVE], _dir[MAXDIR], _ext[MAXEXT];
    int n;

    if (drv && drv[0]) {
        _drv[0] = drv[0];
        _drv[1] = ':';
    } else {
        _drv[0] = 0;
    }
    if (dir && dir[0]) {
        strcpy(_dir, dir);
        n = strlen(_dir);
        if (n && _dir[n-1] != '¥¥' && _dir[n-1] != '/') {
            _dir[n] = '¥¥';
            _dir[n+1] = 0;
        }
    } else {
        _dir[0] = 0;
    }
    if (ext && ext[0]) {
        if (ext[0] != '.') {
            _ext[0] = '.';
            strcpy(_ext+1, ext);
        } else {
            strcpy(_ext, ext);
        }
    } else {
        _ext[0] = 0;
    }
    fnmerge(path, _drv, _dir, file, _ext);
}

/*
 *  環境変数を使ったファイル検索
 *      name: ファイル名
 *      env: 基準となる環境変数
 *      path: 見つかったパス名
 */
void _searchenv(const char *name, const char *env, char *path)
{
    char *pathstr, *envstr, *tmpstr, *findstr;

    /* 元の環境変数 PATH の内容を保存する */
    pathstr = getenv("PATH");
    if (pathstr) {
        pathstr = strdup(pathstr);
        if (pathstr == NULL)
            goto Error2;
    }

    /* 指定された環境変数の内容を取得する */
    if ((envstr = getenv(env)) == NULL)
        goto Error;

    /* 取得した内容を PATH に割り当てて検索させる */
    tmpstr = (char *)malloc(strlen(envstr) + 6);
    if (tmpstr == NULL)
        goto Error;
    strcpy(tmpstr, "PATH=");
    strcat(tmpstr, envstr);
    putenv(tmpstr);
    findstr = searchpath(name);
    if (findstr == NULL)
        *path = 0;
    else
        strcpy(path, findstr);
    free(tmpstr);
    /* 環境変数 PATH の内容を元に戻す */
    tmpstr = (char *)malloc(strlen(pathstr) + 6);
    if (tmpstr == NULL)
        goto Error;
    strcpy(tmpstr, "PATH");
    strcat(tmpstr, pathstr);
    putenv(tmpstr);
    free(tmpstr);
    free(pathstr);
    return;

Error:
    free(pathstr);
Error2:
    *path = 0;
    return;
}

/*
 *  システム情報の取得
 *      bit0: (reserved)
 *      bit1: co-processor
 *      bit2: (reserved)
 *      bit3..7: the number of drives(= 2)
 *      bit8: mouse
 *      bit9..11: the number RS-232C(= 1)
 *      bit12: GP-IB(not supported)
 *      bit13: sound board(not supported)
 *      bit14..15: the number of printer(= 1)
 */
unsigned _bios_equiplist(void)
{
    unsigned ret = 0;

    if (pc98equip() & 0x0030)   /* コプロセッサ */
        ret |= 0x0002;
    ret |= 0x0010;              /* ドライブは2台と仮定 */
    if (pc98mouse(0, NULL))     /* マウス */
        ret |= 0x0100;
    ret |= 0x0200;              /* RS-232C は1ポートと仮定 */
    ret |= 0x4000;              /* プリンタは1台と仮定 */
    return ret;
}

/*
 *  キーボードBIOS
 *      cmd: キーボードBIOSコマンド (_KEYBRD_xxxx)
 *      ktbl: キーボードセンステーブル (_KEYBRD_SENSE のみ)
 *  戻り値
 *  cmd = _KEYBRD_READ (入力)
 *  cmd = _KEYBRD_READY (非破壊入力)
 *      上位バイト (スキャンコード) 、下位バイト (内部コード)
 *  cmd = _KEYBRD_SHIFTSTATUS (シフトステータス)
 *      ビット0～4 : SHIFT, CAPS, カナ, GRPH, CTRL
 *  cmd = _KEYBRD_INITIALIZE (キーボードの初期化)
 *  cmd = _KEYBRD_SENSE (キーボードセンス)
 *      なし (押されているキーボードの状態を ktbl に格納)
 */
unsigned _bios_keybrd(unsigned cmd, unsigned char *ktbl)
{
    if (cmd == _KEYBRD_SENSE) {
        int grp;

        for (grp = 0; grp < 16; grp++)
            ktbl[grp] = pc98key(_KEYBRD_SENSE, grp);
        return 0;
    }
    return pc98key(cmd);
}

/*
 *  プリンタBIOS
 *      cmd: プリンタBIOSコマンド (_PRN_xxx)
 *      data: データ文字 (文字列)
 *  戻り値
 *  cmd = _PRN_INIT (プリンタの初期化)
 *  cmd = _PRN_WRITE (プリンタへ1バイト出力)
 *  cmd = _PRN_STATUS (プリンタステータスの取得)
 *      0 : 未出力、1 : 正常終了、2 : タイムアウト
 *  cmd = _PRN_STRING (プリンタへ文字列出力)
 *      未出力のバイト数
 */
unsigned _bios_printer(unsigned cmd, unsigned char *data)
{
    if (cmd == _PRN_STRING)
        return pc98print(cmd, strlen((char *)data), data);
    return (unsigned)pc98print(cmd, *data) >> 8;
}

static struct REGPACK regs;
#define RAX regs.r_ax
#define RBX regs.r_bx
#define RCX regs.r_cx
#define RDX regs.r_dx
#define RSI regs.r_si
#define RDI regs.r_di
#define RBP regs.r_bp
#define RDS regs.r_ds
#define RES regs.r_es
#define RCY (regs.r_flags & 1)
#define DOS(cmd) { RAX = (cmd) << 8; intr(0x21, &regs); }

/*
 *  カレンダBIOS
 *      cmd: カレンダBIOSコマンド
 *          cmd = _TIME_GETCLOCK (日付・時刻の取得)
 *          cmd = _TIME_SETCLOCK (日付・時刻の設定)
 *      buf: 日付・時刻の内容
 *          buf[0] : 年(00～0x99) ... BCD
 *          buf[1] : 月(0～0xC) | 曜(0～6) ... 4ビット16進×2
 *          buf[2] : 日(0～0x31) ... BCD
 *          buf[3] : 時(0～0x23) ... BCD
 *          buf[4] : 分(0～0x59) ... BCD
 *          buf[5] : 秒(0～0x59) ... BCD
 */
unsigned _bios_timeofday(unsigned cmd, char *buf)
{
    RES = FP_SEG(buf);
    RBX = FP_OFF(buf);
    RAX = cmd << 8;
    intr(0x1C, &regs);
    return 0;
}

/*
 *  次の割り込みハンドラへの移行
 */
#pragma argsused
void _chain_intr(void (interrupt far *_handler)())
{
    _asm {
        mov sp, bp
        pop bp
        pop ax      /* 呼び出した関数への戻りアドレスを破棄 */
    }
#if defined(__MEDIUM__) || defined(__LARGE__) || defined(__HUGE__)
    _asm pop    ax  /* 呼び出した関数への戻りアドレス (セグメント) を破棄 */
#endif
    _asm {
        mov bx, [bp+14]
        mov ax, [bp+16]
        pop WORD PTR [bp+14]
        pop WORD PTR [bp+16]
        mov sp, bp
        pop bp
        pop di
        pop si
        pop ds
        pop es
        pop dx
        pop cx
        cli
        retf
    }
}
```

Cプログラミング用語

プロトタイプ　関数定義時に行う，引数の型宣言のこと．ANSI Cではこのプロトタイプ宣言で引数の型をチェックする．

```c

/*
 *  DOSファンクションの呼び出し
 *      戻り値は、0で正常終了を、0以外はDOSファンクションの
 *      エラーコードをあらわします。
 */

/*
 *  DOSのエラーコードから errno を設定する関数
 */
extern char _dosErrorToSV[];/* 0～88までのエラー番号テーブル（非公開変数）*/
extern int _doserrno;
extern int errno;
static int _set_doserror(int errcode)
{
    _doserrno = errcode;
    if (0 <= errcode && errcode <= 88)
        errno = _dosErrorToSV[errcode];
    return errcode;
}

#define CY_RET  { if (RCY) return _set_doserror(RAX); }
#define RETVAL  (RCY ? _set_doserror(RAX) : 0)

/*
 *  メモリブロックの割り当て
 *      size: 割り当てるブロックのパラグラフサイズ
 *      segp: 割り当てられたメモリブロックのセグメント
 *          （エラーの場合は割り当て可能な最大メモリサイズ）
 */
unsigned _dos_allocmem(unsigned size, unsigned *segp)
{
    RBX = size;
    DOS(0x48);
    if (RCY) {
        *segp = RBX;
        return _set_doserror(RAX);
    }
    *segp = RAX;
    return 0;
}

/*
 *  ファイルハンドルのクローズ
 *      handle: ファイルハンドル
 */
unsigned _dos_close(int handle)
{
    RBX = handle;
    DOS(0x3e);
    return RETVAL;
}

/*
 *  ファイルの作成
 *      path: ファイル名
 *      attr: ファイルの属性（_A_NORMAL, _A_RDONLY, _A_HIDDEN, _A_SYSTEM）
 *      handle: オープンされたファイルハンドル
 */
unsigned _dos_creat(const char *path, unsigned attr, int *handle)
{
    RDS = FP_SEG(path);
    RDX = FP_OFF(path);
    RCX = attr;
    DOS(0x3C);
    CY_RET;
    *handle = RAX;
    return 0;
}

/*
 *  ファイルの新規作成（ファイルが存在していればエラー）
 *      path: ファイル名
 *      attr: ファイルの属性（_A_NORMAL, _A_RDONLY, _A_HIDDEN, _A_SYSTEM）
 *      handle: オープンされたファイルハンドル
 */
unsigned _dos_creatnew(const char *path, unsigned attr, int *handle)
{
    RDS = FP_SEG(path);
    RDX = FP_OFF(path);
    RCX = attr;
    DOS(0x5B);
    CY_RET;
    *handle = RAX;
    return 0;
}

/*
 *  メモリブロックの解放
 *      segp: 解放するメモリブロックのセグメント値
 */
unsigned _dos_freemem(unsigned segp)
{
    RES = segp;
    DOS(0x49);
    return RETVAL;
}

/*
 *  日付の取得
 *      datep: 日付をあらわす構造体へのポインタ
 */
void _dos_getdate(struct dosdate_t *datep)
{
    DOS(0x2A);
    datep->dayofweek = (unsigned char)RAX; /* 0(日曜)～6 */
    datep->day = (unsigned char)RDX;    /* 1～31 */
    datep->month = RDX >> 8;            /* 1～12 */
    datep->year = RCX;                  /* 1980～2099 */
}

/*
 *  カレントドライブの取得
 *      drive: 現在のカレントドライブ（1=A:, 2=B:, ...）
 */
void _dos_getdrive(unsigned *drive)
{
    DOS(0x19);
    *drive = (unsigned char)RAX + 1;
}

/*
 *  ディスクの空き容量の取得
 *      drive: 調べるドライブ（1=A:, 2=B:, ...）
 *      dinfo: 空き容量をあらわす構造体へのポインタ
 */
unsigned _dos_getdiskfree(unsigned drive, struct diskfree_t *dinfo)
{
    RDX = drive;
    DOS(0x36);
    if (RAX == 0xFFFFu)
        return 22;
    dinfo->total_clusters = RDX;        /* 総クラスタ数 */
    dinfo->avail_clusters = RBX;        /* 空きクラスタ数 */
    dinfo->sectors_per_cluster = RAX;   /* クラスタごとのセクタ数 */
    dinfo->bytes_per_sector = RCX;      /* セクタごとのバイト数 */
    return 0;
}

/*
 *  ファイルの属性の取得
 *      file: ファイル名
 *      attr: 取得したファイルの属性
 *          （_A_NORMAL, _A_RDONLY, _A_HIDDEN, _A_SYSTEM
 *            _A_VOLID, _A_SUBDIR, _A_ARCH）
 */
unsigned _dos_getfileattr(const char *file, unsigned *attr)
{
    RDS = FP_SEG(file);
    RDX = FP_OFF(file);
    DOS(0x43);
    CY_RET;
    *attr = RCX;
    return 0;
}

/*
 *  ファイルのタイムスタンプの取得
 *      handle: オープンされたファイルハンドル
 *      datep: 取得した日付情報
 *              （7ビット／4ビット／5ビット＝年／月／日）
 *      timep: 取得した時刻情報
 *              （5ビット／6ビット／5ビット＝時／分／（秒÷2））
 */
unsigned _dos_getftime(int handle, unsigned *datep, unsigned *timep)
{
    RAX = 0x5700;
    RBX = handle;
    intr(0x21, &regs);
    CY_RET;
    *datep = RDX;
    *timep = RCX;
    return 0;
}

/*
 *  時刻の取得
 *      timep: 時刻をあらわす構造体へのポインタ
 */
void _dos_gettime(struct dostime_t *timep)
{
    DOS(0x2C);
    timep->hour = RCX >> 8;
    timep->minute = (unsigned char)RCX;
    timep->second = RDX >> 8;
    timep->hsecond = (unsigned char)RDX;
}

/*
 *  ファイルのオープン
 *      path: ファイル名
 *      flags: オープンモード（O_RDONLY, O_WRONLY, O_RDWR, SH_xxxx）
 *      handle: オープンされたファイルハンドル
 *  ※flags に直接数値を指定してはいけない。
 */
unsigned _dos_open(const char *path, unsigned flags, int *handle)
{
    int dosflags;
    if (flags & O_WRONLY)
        dosflags = 1;
    else if (flags & O_RDWR)
        dosflags = 2;
    else
        dosflags = 0;
    dosflags |= flags & 0xF0;
    RAX = 0x3D00 + dosflags;
    RDS = FP_SEG(path);
    RDX = FP_OFF(path);
    intr(0x21, &regs);
    CY_RET;
    *handle = RAX;
    return 0;
}

/*
 *  データの読み込み
 *      handle: ファイルハンドル
 *      buf: データを読み込むためのバッファ
 *      len: 読み込むバイト数
 *      nread: 実際に読み込まれたバイト数
 */
unsigned _dos_read(int handle, void _far *buf, unsigned len, unsigned *nread)
{
    RBX = handle;
    RCX = len;
    RDS = FP_SEG(buf);
```


Cプログラミング用語
分割コンパイル　1つのプログラムを複数のソースファイルで作成し，別々にコンパイルした後，実行ファイル作成時リンクすること．

```
    RDX = FP_OFF(buf);
    DOS(0x3F);
    CY_RET;
    *nread = RAX;
    return 0;
}

/*
 *  メモリブロックの大きさの変更
 *      size: 変更する大きさ
 *      segx: メモリブロックのセグメント値
 *      maxp: 変更できなかった場合は、使用可能な最大メモリサイズ
 */
unsigned _dos_setblock(unsigned size, unsigned segx, unsigned *maxp)
{
    RES = segx;
    RBX = size;
    DOS(0x4A);
    if (RCY) {
        *maxp = RBX;
        return _set_doserror(RAX);
    }
    return 0;
}

/*
 *  日付の設定
 *      datep: 日付をあらわす構造体へのポインタ
 *  戻り値
 *      0 (成功) 、0xFFFF (不正な日付)
 */
unsigned _dos_setdate(struct dosdate_t *datep)
{
    RDX = (datep->month << 8) | datep->day;
    RCX = datep->year;
    DOS(0x2B);
    if ((unsigned char)RAX == (unsigned char)0xFF)
        return 0xFFFFu;
    return 0;
}

/*
 *  カレントドライブの設定
 *      drive: ドライブ番号 (1=A:, 2=B:, ...)
 *      ndrives: 現在接続されているドライブ数
 */
void _dos_setdrive(unsigned drive, unsigned *ndrives)
{
    RDX = drive - 1;
    DOS(0x0E);
    *ndrives = (unsigned char)RAX;
}

/*
 *  ファイルの属性の設定
 *      file: ファイル名
 *      attr: 設定するファイルの属性
 *          (_A_NORMAL, _A_RDONLY, _A_HIDDEN, _A_SYSTEM,
 *           _A_VOLID, _A_SUBDIR, _A_ARCH)
 */
unsigned _dos_setfileattr(const char *file, unsigned attr)
{
    RDS = FP_SEG(file);
    RDX = FP_OFF(file);
    RCX = attr;
    RAX = 0x4301;
    intr(0x21, &regs);
    return RETVAL;
}

/*
 *  ファイルのタイムスタンプの設定
 *      handle: オープンされたファイルハンドル
 *      datep: 設定する日付情報
 *              (7ビット/4ビット/5ビット=年/月/日)
 *      timep: 設定する時刻情報
 *              (5ビット/6ビット/5ビット=時/分/(秒÷2))
 */
unsigned _dos_setftime(int handle, unsigned date, unsigned time)
{
    RBX = handle;
    RCX = time;
    RDX = date;
    RAX = 0x5701;
    intr(0x21, &regs);
    return RETVAL;
}

/*
 *  時刻の設定
 *      timep: 時刻をあらわす構造体へのポインタ
 */
unsigned _dos_settime(struct dostime_t *timep)
{
    RCX = (timep->hour << 8) | timep->minute;
    RDX = (timep->second << 8) | timep->hsecond;
    DOS(0x2D);
    if ((unsigned char)RAX == (unsigned char)0xFF)
        return 0xFFFFu;
    return 0;
}

/*
 *  データの書き込み
 *      handle: ファイルハンドル
 *      buf: 書き込むデータのあるバッファ
 *      len: 書き込むバイト数
 *      nread: 実際に書き込まれたバイト数
 */
unsigned _dos_write(int handle, const void _far *buf, unsigned len,
        unsigned *nwrite)
{
    RBX = handle;
    RCX = len;
    RDS = FP_SEG(buf);
    RDX = FP_OFF(buf);
    DOS(0x40);
    CY_RET;
    *nwrite = RAX;
    return 0;
}

static void (far *hfunc)(unsigned derr, unsigned ecode, unsigned far *devhdr);
static int hardfunc(unsigned di, unsigned ax, unsigned bp, unsigned si)
{
    hfunc(di, ax, MK_FP(bp, si));
    return 0;
}

/*
 *  致命的エラーハンドラ
 *      handler: エラーハンドラへのポインタ
 */
void _harderr(void (far *handler)())
{
    hfunc = (void (far *)(unsigned,unsigned,unsigned far *))handler;
    harderr(hardfunc);
}

/*
 *  現在の日付の文字列化
 *      str: 文字列化した内容を格納するバッファ
 */
char *_strdate(char *str)
{
    struct dosdate_t d;

    _dos_getdate(&d);
    sprintf(str, "%02d/%02d/%02d", d.year - 1900, d.month, d.day);
    return str;
}

/*
 *  現在の時刻の文字列化
 *      str: 文字列化した内容を格納するバッファ
 */
char *_strtime(char *str)
{
    struct dostime_t t;

    _dos_gettime(&t);
    sprintf(str, "%02d:%02d:%02d", t.hour, t.minute, t.second);
    return str;
}

#else /* defined(__TURBOC__) */
/*
 *  Microsoft C のための互換ルーチン
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <jstring.h>
#include <stdarg.h>
#include <time.h>
#include <dos.h>
#include <conio.h>
#include "dir.h"
#include "pc98.h"
#include "dual.h"

#pragma check_stack(off)

/*
 *  割り込み処理
 *      intno: 割り込み番号
 *      regs: パックレジスタ構造体 (REGPACK) へのポインタ
 *            入出力用に使われる (BP を返すことはできない)
 */
void intr(int intno, struct REGPACK *regs)
{
    void (far *vect)(void);
    char code[6];
    unsigned bx_, es_;

    vect = (void (far *)(void))(void far *)code;
    code[0] = 0xBD;     /* MOV BX,???? */
    *(unsigned *)(code+1) = regs->r_bp;
    code[3] = 0xCD;     /* INT intno */
    code[4] = (char)intno;
    code[5] = 0xCB;     /* RETF */

    es_ = FP_SEG(regs);
    bx_ = FP_OFF(regs);

    _asm {
        push ds
        mov bx, [bx_]
        mov es, [es_]
        push bx
        push es
        push WORD PTR es:[bx+2]     /* BX */
        push WORD PTR es:[bx+16]    /* ES */
        mov ax, es:[bx]
        mov cx, es:[bx+4]
        mov dx, es:[bx+6]
        mov si, es:[bx+10]
        mov di, es:[bx+12]
        mov ds, es:[bx+14]
        push WORD PTR es:[bx+18]
        popf
        pop es
        pop bx
        push bp
    }
```

Cプログラミング用語
ヘッダファイル　ソースプログラムに関連した関数の集合，定数の集合宣言などヘッダ部分を独立したファイル．

```
    (*vect)();
    _asm {
        pop bp
        mov [bx_], bx
        mov [es_], es
        pop es
        pop bx
        pushf
        pop WORD PTR es:[bx+18]
        mov es:[bx+14], ds
        mov es:[bx+12], di
        mov es:[bx+10], si
        mov es:[bx+6], dx
        mov es:[bx+4], cx
        mov es:[bx], ax
        mov ax, [bx_]
        mov es:[bx+2], ax
        mov ax, [es_]
        mov es:[bx+16], ax
        pop ds
    }
}

static struct REGPACK regs;
#define RAX regs.r_ax
#define RBX regs.r_bx
#define RCX regs.r_cx
#define RDX regs.r_dx
#define RSI regs.r_si
#define RDI regs.r_di
#define RBP regs.r_bp
#define RDS regs.r_ds
#define RES regs.r_es
#define RFLAGS regs.r_flags
#define RCY (regs.r_flags & 1)
#define DOS(cmd) { RAX = (cmd) << 8; intr(0x21, &regs); }

/*
 *  Borland C++ の疑似レジスタ変数と geninterrupt を関数でまねる
 *  インライン展開されるわけではないので、割り込みハンドラなどで
 *  使用することはできない。
 */

struct __tag_pseudo_regs __pseudo_regs;
/*unsigned _AX, _BX, _CX, _DX, _SI, _DI, _BP, _DS, _ES, _FLAGS;*/
void geninterrupt(int intno)
{
    RAX = _AX;
    RBX = _BX;
    RCX = _CX;
    RDX = _DX;
    RSI = _SI;
    RDI = _DI;
    RBP = _BP;
    RDS = _DS;
    RES = _ES;
    RFLAGS = _FLAGS;
    intr(intno, &regs);
    _AX = RAX;
    _BX = RBX;
    _CX = RCX;
    _DX = RDX;
    _SI = RSI;
    _DI = RDI;
    _BP = RBP;
    _DS = RDS;
    _ES = RES;
    _FLAGS = RFLAGS;
}

/*
 *  カレントディレクトリの取得
 *      drive: ドライブ番号 (0=カレントドライブ、1=A:, 2=B:, ...)
 *      dir: 取得したディレクトリ
 */
int getcurdir(int drive, char *dir)
{
    RDX = drive;
    RDS = FP_SEG(dir);
    RSI = FP_OFF(dir);
    DOS(0x47);
    if (RCY) return -1;
    return 0;
}

/*
 *  環境変数 PATH によるファイルの検索
 *      file: 検索するファイル
 *  戻り値
 *      見つかったパス名へのポインタ
 */
char *searchpath(const char *file)
{
    static char FoundFile[MAXPATH];

    _searchenv(file, "PATH", FoundFile);
    return FoundFile;
}

/*
 *  メモリブロックの割り当て
 *      size: 割り当てるブロックのパラグラフサイズ
 *      segp: 割り当てられたメモリブロックのセグメント
 *  戻り値
 *      -1 (成功) 、-1 以外 (割り当て可能な最大メモリサイズ)
 */
int allocmem(unsigned size, unsigned *segp)
{
    RBX = size;
    DOS(0x48);
    if (RCY) return RBX;
    *segp = RAX;
    return -1;
}

/*
 *  DOS ファンクションの呼び出し
 *      dosfun: AH レジスタに設定する値
 *      ptr: DS:DX に設定する値
 *      dosal: AL レジスタに設定する値
 *  戻り値
 *      システムコール呼び出し後の AX レジスタの内容
 */
int bdosptr(int dosfun, void *ptr, unsigned dosal)
{
    RAX = (dosfun << 8) | (unsigned char)dosal;
    RDS = FP_SEG(ptr);
    RDX = FP_OFF(ptr);
    intr(0x21, &regs);
    if (RCY)
        return -1;
    return RAX;
}

static int (*brkptr)(void) = NULL;
static void interrupt far brkhandler(void)
{
    if (brkptr && !(*brkptr)())
        _exit(0);
}

/*
 *  CTRL-C 割り込みハンドラの設定
 *      handler: 割り込みハンドラへのポインタ
 */
void ctrlbrk(int (*handler)(void))
{
    brkptr = handler;
    _dos_setvect(0x23, brkhandler);
}

/*
 *  時間待ち
 *      msecs: 待つ時間 (ミリ秒単位)
 *  ※ハイレゾリューションモードでは正しい時間になりません。
 */
void delay(unsigned msecs)
{
    unsigned long cnt;

    /* 高解像度ディスプレイの水平同期信号 (24.83Hz) を利用 */
    cnt = (long)msecs * 2483 / 100;
    while (cnt--) {
        while (inportb(0x60) & 0x40)
            ;
        while (!(inportb(0x60) & 0x40))
            ;
    }
}

/*
 *  日付の取得
 *      datep: 日付をあらわす構造体へのポインタ
 */
void getdate(struct date *datep)
{
    DOS(0x2C);
    ((unsigned *)datep)[0] = RCX;
    ((unsigned *)datep)[1] = RDX;
}

/*
 *  ディスクの空き容量の取得
 *      drive: 調べるドライブ (0=カレントドライブ, 1=A:, 2=B:, ...)
 *      dtable: 空き容量をあらわす構造体へのポインタ
 */
void getdfree(unsigned char drive, struct dfree *dtable)
{
    RDX = drive;
    DOS(0x36);
    ((unsigned *)dtable)[0] = RBX;
    ((unsigned *)dtable)[1] = RDX;
    ((unsigned *)dtable)[2] = RCX;
    ((unsigned *)dtable)[3] = RAX;
}

/*
 *  FAT 情報の取得
 *      drive: 調べるドライブ (0=カレントドライブ, 1=A:, 2=B:, ...)
 *      dtable: FAT 情報をあらわす構造体へのポインタ
 */
void getfat(unsigned char drive, struct fatinfo *dtable)
{
    RDX = drive;
    DOS(0x1C);
    ((unsigned char *)dtable)[0] = (unsigned char)RAX;
    ((unsigned char *)dtable)[1] = *(unsigned char far *)MK_FP(RDS, RBX);
    ((unsigned *)dtable)[1] = RDX;
    ((unsigned *)dtable)[2] = RCX;
}

/*
 *  PSP の取得
 */
unsigned getpsp(void)
{
    DOS(0x62);
    return RBX;
}

/*
 *  スイッチ文字の取得
 */
int getswitchar(void)
{
    RAX = 0x3700;
```


C プログラミング用語
変数型　変数がメモリ空間を占める形式の種類

```c
    intr(0x21, &regs);
    return (unsigned char)RDX;
}

/*
 *  時刻の取得
 *      timep: 時刻をあらわす構造体へのポインタ
 */
void gettime(struct time *timep)
{
    DOS(0x2C);
    ((unsigned *)timep)[0] = RCX;
    ((unsigned *)timep)[1] = RDX;
}

/*
 *  ベリファイフラグの取得
 *  戻り値
 *      0 (ベリファイオフ)、1 (ベリファイオン)
 */
int getverify(void)
{
    DOS(0x54);
    return (int)(unsigned char)RAX;
}

static int (*hfunc)(unsigned di, unsigned ax, unsigned bp, unsigned si);
static void far hardfunc(unsigned deverr, unsigned errcode,
            unsigned far *devhdr)
{
    hfunc(deverr, errcode, FP_SEG(devhdr), FP_OFF(devhdr));
}

/*
 *  致命的エラーハンドラの設定
 *      handler: エラーハンドラへのポインタ
 */
void harderr(int (*handler)())
{
    hfunc = (int (*)(unsigned,unsigned,unsigned,unsigned))handler;
    _harderr(hardfunc);
}

/*
 *  ビープ音の停止
 */
void nosound(void)
{
    int n;
    if (inportb(0x42) & 0x20)
        n = 1229;
    else
        n = 998;
    outp(0x3FDF, 0x7E);
    outp(0x3FDB, (unsigned char)n);
    outp(0x3FDB, (unsigned char)(n >> 8));
    outp(0x37, 0x07);
}

/*
 *  メモリブロックの大きさの変更
 *      segx: メモリブロックのセグメント値
 *      size: 変更する大きさ
 *  戻り値
 *      -1 (成功)、-1 以外 (使用可能な最大メモリサイズ)
 */
int setblock(unsigned segx, unsigned size)
{
    RBX = size;
    RES = segx;
    DOS(0x4A);
    if (RCY) return RBX;
    return -1;
}

/*
 *  日付の設定
 *      datep: 日付をあらわす構造体へのポインタ
 */
void setdate(struct date *datep)
{
    RCX = ((unsigned *)datep)[0];
    RDX = ((unsigned *)datep)[1];
    DOS(0x2B);
}

/*
 *  スイッチ文字の設定
 *      ch: スイッチ文字コード
 */
void setswitchar(char ch)
{
    RDX = ch;
    RAX = 0x3701;
    intr(0x21, &regs);
}

/*
 *  時刻の設定
 *      timep: 時刻をあらわす構造体へのポインタ
 */
void settime(struct time *timep)
{
    RCX = ((unsigned *)timep)[0];
    RDX = ((unsigned *)timep)[1];
    DOS(0x2D);
}

/*
 *  ベリファイフラグの設定
 *      value: 0 (ベリファイオフ)、1 (ベリファイオン)
 */
void setverify(int value)
{
    RAX = 0x2E00 + (unsigned char)value;
    intr(0x21, &regs);
}

/*
 *  実行の一時停止
 *      sec: 停止する秒数
 */
void sleep(unsigned sec)
{
    long t1, t2;

    time(&t1);
    do {
        time(&t2);
    } while (t1 == t2);
    while (sec) {
        t1 = t2;
        do {
            time(&t2);
        } while (t1 == t2);
        sec--;
    }
}

/*
 *  ビープ音の設定/鳴動
 *      freq: 周波数(Hz)
 */
void sound(unsigned freq)
{
    long F;
    long n;

    if (inportb(0x42) & 0x20)
        F = 2457600L;   /* 5/10/12/20MHz */
    else
        F = 1996800L;   /* 8/16/33MHz */
    n = F / freq;
    if (n > 0xFFFFu) n = 0xFFFF;
    outp(0x3FDF, 0x7E);
    outp(0x3FDB, (unsigned char)n);
    outp(0x3FDB, (unsigned char)(n >> 8));
    outp(0x37, 0x06);
}

/*
 *  DTA の取得
 */
char far *getdta(void)
{
    DOS(0x2F);
    return (char far *)MK_FP(RES, RBX);
}

/*
 *  DTA の設定
 *      dta: ディスク転送アドレス
 */
void setdta(char far *dta)
{
    RDS = FP_SEG(dta);
    RDX = FP_OFF(dta);
    DOS(0x1A);
}

/*
 *  ＢＩＯＳ関数
 */

/*
 *  CRT BIOS の呼び出し
 *      cmd については BIOS の解説書を参考にしてください
 */
int pc98crt(int cmd, ...)
{
    struct SREGS sregs;
    va_list ap;

    segread(&sregs);
    va_start(ap, cmd);
    RDX = va_arg(ap, unsigned);
    RCX = va_arg(ap, unsigned);
#if (defined(M_I86SM) || defined(M_I86MM))
    RBX = va_arg(ap, unsigned);
#else
    RBX = sregs.ds;
#endif
    va_end(ap);
    return RAX;
}

/*
 *  キーボードＢＩＯＳ
 *      cmd については _bios_keybrd を参考にしてください
 */
unsigned pc98key(int cmd, ...)
{
    unsigned char ktbl[16];
    int grp;
    va_list ap;

    if (cmd == _KEYBRD_SENSE) {
        va_start(ap, cmd);
        grp = va_arg(ap, int);
        _bios_keybrd(cmd, ktbl);
        va_end(ap);
        return ktbl[grp];
    }
    return _bios_keybrd(cmd, ktbl);
}

```

Cプログラミング用語
ポインタ　間接的に各種変数を参照するオブジェクトのこと．文字変数などに利用され，変数自体はその格納アドレスを指す．

```c
static void (*mouse_fptr)(unsigned, unsigned, unsigned, unsigned);
static void far _loadds mouse_handler(void)
{
    unsigned ax_, bx_, cx_, dx_;

    _asm {
        cld
        mov [ax_], ax
        mov [bx_], bx
        mov [cx_], cx
        mov [dx_], dx
    }
    (*mouse_fptr)(ax_, bx_, cx_, dx_);
}

/*
 * マウス BIOS の呼び出し
 * ※cmd については MS-DOS のプログラマーズガイドを参考にしてください
 */
int pc98mouse(int cmd, struct mouseinfo *mp)
{
    if (mp) {
        RAX = cmd;
        RBX = mp->reg_bx;
        RCX = mp->reg_cx;
        RDX = mp->reg_dx;
        if (cmd == 9) { /* set a user-defined handler */
            RES = FP_SEG(mp->shape);
            RDX = FP_OFF(mp->shape);
        } else if (cmd == 12) {
            mouse_fptr = mp->fptr;
            RES = FP_SEG(mouse_handler);
            RDX = FP_OFF(mouse_handler);
        }
    }
    intr(0x33, &regs);
    if (mp) {
        mp->reg_ax = RAX;
        mp->reg_bx = RBX;
        mp->reg_cx = RCX;
        mp->reg_dx = RDX;
    }
    return RAX;
}

/*
 * プリンタ BIOS の呼び出し
 *     cmd については _bios_printer を参考にしてください
 */
int pc98print(int cmd, ...)
{
    va_list ap;
    int len;
    char *buf;

    va_start(ap, cmd);
    len = va_arg(ap, int);
    buf = va_arg(ap, char *);
    va_end(ap);
    RAX = (cmd << 8) + (unsigned char)len;
    RCX = len;
    RES = FP_SEG(buf);
    RBX = FP_OFF(buf);
    intr(0x1A, &regs);
    return (cmd == 0x30) ? RCX : RAX;
}

static void (*timerfunc)(void) = NULL;
static volatile int timerdone = 1;

static void interrupt far timer_handler()
{
    void (*tfunc)(void) = timerfunc;

    timerfunc = NULL;
    timerdone = 1;
    if (tfunc) (*tfunc)();
}

/*
 * タイマー BIOS の呼び出し
 *     interval: 割り込みがかかるまでの期間 (10ms単位)
 *     fptr: 割り込みハンドラへのポインタ
 */
void (*pc98timer(unsigned interval, void (*fptr)()))()
{
    union REGS regs;
    struct SREGS sregs;
    void (*prevfunc)() = timerfunc;

    timerdone = 0;
    timerfunc = fptr;
    if (fptr == NULL) {
        regs.x.cx = 1;
    } else {
        timerfunc = fptr;
        regs.x.cx = interval;
    }
    regs.x.bx = FP_OFF(timer_handler);
    sregs.es = FP_SEG(timer_handler);
    regs.h.ah = 0x02;
    int86x(0x1c, &regs, &regs, &sregs);
    if (fptr == NULL) {
        while (!timerdone)
            ;
    }
    return prevfunc;
}

/*
 * システムフォントの取得
 *     code: 文字コード (0x80xx=ANK, SHIFT-JIS, JIS)
 *     buffer: 文字フォントバッファ
 * ※フォントの形式は、_getkanjiを参考にしてください
 */
void pc98getfont(unsigned code, void *buffer)
{
    if (code >= 0x8100)
        code = jmstojis(code);
    RDX = code;
    RBX = FP_SEG(buffer);
    RCX = FP_OFF(buffer);
    RAX = 0x1400;
    intr(0x18, &regs);
}

/*
 * 外字の定義
 *     code: 文字コード (0x80xx=ANK, SHIFT-JIS, JIS)
 *     buffer: 文字フォントバッファ
 * ※フォントの形式は、_getkanjiを参考にしてください
 */
void pc98putuserfont(unsigned code, void *buffer)
{
    if (code >= 0x8100)
        code = jmstojis(code);
    RDX = code;
    RBX = FP_SEG(buffer);
    RCX = FP_OFF(buffer);
    RAX = 0x1A00;
    intr(0x18, &regs);
}

static void __dputs(const char *str)
{
    while (*str) {
        RCX = 0x10;
        RAX = 0;
        RDX = *str;
        intr(0xDC, &regs);
        str++;
    }
}

/*
 * ファンクションキーの表示
 */
void pc98fkeyon(void)
{
    __dputs("¥033[>1l");
}

/*
 * ファンクションキーの消去
 */
void pc98fkeyoff(void)
{
    __dputs("¥033[>1h");
}

/*
 * ファンクションキーが表示されているかどうかを調べる
 * 戻り値
 *     0 (表示されていない) 、1 (表示されている)
 */
int pc98fkeychk(void)
{
    return (*(unsigned char far *)MK_FP(0, 0x0711)) != 0;
}

/*
 * ファンクションキーの内容の取得
 *     id: ファンクションキー番号
 *         0 = すべて
 *         1..10 = [f·1] ～ [f·10]
 *         11..20 = [SHIFT]+[f·1] ～ [SHIFT]+[f·10]
 *         21..31 = [ROLLUP], [ROLLDOWN], [INS], [DEL],
 *                  [↑], [←], [→], [↓], [HOME], [HELP], [SHIFT]+[HOME]
 *     buffer: ファンクションキーの内容
 * ※詳しくは MS-DOS プログラマーズガイドを参考にしてください
 */
void pc98getfkey(int id, void *buffer)
{
    RCX = 0x0C;
    RAX = id;
    RDS = FP_SEG(buffer);
    RDX = FP_OFF(buffer);
    intr(0xDC, &regs);
}

/*
 * ファンクションキーの内容の設定
 *     id: ファンクションキー番号 (pc98getfkeyと同じ)
 *     buffer: ファンクションキーの内容
 * ※詳しくは MS-DOS プログラマーズガイドを参考にしてください
 */
void pc98setfkey(int id, void *buffer)
{
    RCX = 0x0D;
    RAX = id;
    RDS = FP_SEG(buffer);
    RDX = FP_OFF(buffer);
    intr(0xDC, &regs);
}

/*
 * ファイルのアクセスモードの変更
 *     path: ファイル名
 *     func: 0 (取得) 、1 (設定)
 *     attr: 属性
 *         (FA_RDONLY, FA_HIDDEN, FA_SYSTEM, FA_LABEL, FA_DIREC, FA_ARCH)
 */
int _chmod(const char *path, int func, ... /* attr */)
{
    va_list ap;
    int attr;
```


C プログラミング用語

前判定反復文　制御文の一種で，while～文のように継続条件を前で判断する文．

```

    va_start(ap, func);
    attr = va_arg(ap, int);
    va_end(ap);
    RCX = attr;
    RAX = 0x4300 + (unsigned char)func;
    RDS = FP_SEG(path);
    RDX = FP_OFF(path);
    intr(0x21, &regs);
    if (RCY) return -1;
    return RCX;
}

/*
 *  ファイルハンドルのクローズ
 *      handle: ファイルハンドル
 */
int _close(int handle)
{
    RBX = handle;
    DOS(0x3E);
    return RCY ? -1 : 0;
}

/*
 *  ファイルの作成
 *      path: ファイル名
 *      attr: ファイルの属性 (_A_NORMAL, _A_RDONLY, _A_HIDDEN, _A_SYSTEM)
 *  戻り値
 *      -1 以外 (ファイルハンドル) 、-1 (エラー)
 */
int _creat(const char *path, int attr)
{
    RCX = attr;
    RDS = FP_SEG(path);
    RDX = FP_OFF(path);
    DOS(0x3C);
    return RCY ? -1 : RAX;
}

/*
 *  ファイルの新規作成 (ファイルが存在していればエラー)
 *      path: ファイル名
 *      attr: ファイルの属性 (FA_RDONLY, FA_HIDDEN, FA_SYSTEM)
 *  戻り値
 *      -1 以外 (ファイルハンドル) 、-1 (エラー)
 */
int creatnew(const char *path, int attr)
{
    RCX = attr;
    RDS = FP_SEG(path);
    RDX = FP_OFF(path);
    DOS(0x5B);
    return RCY ? -1 : RAX;
}

/*
 *  一時ファイルの作成
 *      path: 13バイトの空き領域が続くASCIZ文字列
 *      attr: ファイルの属性 (FA_RDONLY, FA_HIDDEN, FA_SYSTEM)
 */
int creattemp(char *path, int attr)
{
    RCX = attr;
    RDS = FP_SEG(path);
    RDX = FP_OFF(path);
    DOS(0x5A);
    return RCY ? -1 : RAX;
}

/*
 *  ファイルのタイムスタンプの取得
 *      handle: オープンされたファイルハンドル
 *      ftimep: タイムスタンプをあらわす構造体へのポインタ
 */
int getftime(int handle, struct ftime *ftimep)
{
    RAX = 0x5700;
    RBX = handle;
    intr(0x21, &regs);
    if (RCY) return -1;
    ((unsigned *)ftimep)[0] = RCX;
    ((unsigned *)ftimep)[1] = RDX;
    return 0;
}

/*
 *  IOCTL 制御
 *      handle: ハンドル
 *      func: コード
 */
int ioctl(int handle, int func, ... /* ptr, count */ )
{
    va_list ap;
    void *ptr;
    int count;

    va_start(ap, func);
    ptr = va_arg(ap, void *);
    count = va_arg(ap, int);
    va_end(ap);
    RAX = 0x4400 + (unsigned char)func;
    RBX = handle;
    RCX = count;
    RDS = FP_SEG(ptr);
    RDX = FP_OFF(ptr);
    intr(0x21, &regs);
    if (RCY) return -1;
    if (func == 0) return RDX;
    return RAX;
}

/*
 *  ファイルのオープン
 *      path: ファイル名
 *      oflags: オープンモード (O_RDONLY, O_WRONLY, O_RDWR, SH_xxxx)
 *  戻り値
 *      -1 以外 (ファイルハンドル) 、-1 (エラー)
 */
int _open(const char *path, int oflags)
{
    RDS = FP_SEG(path);
    RDX = FP_OFF(path);
    RAX = 0x3D00 + (unsigned char)oflags;
    intr(0x21, &regs);
    if (RCY) return -1;
    return RAX;
}

/*
 *  データの読み込み
 *      handle: ファイルハンドル
 *      buf: データを読み込むためのバッファ
 *      len: 読み込むバイト数
 *  戻り値
 *      実際に読み込まれたバイト数
 */
int _read(int handle, void *buf, unsigned len)
{
    RBX = handle;
    RCX = len;
    RDS = FP_SEG(buf);
    RDX = FP_OFF(buf);
    DOS(0x3F);
    return RCY ? -1 : RAX;
}

/*
 *  ファイルのタイムスタンプの設定
 *      handle: オープンされたファイルハンドル
 *      ftimep: タイムスタンプをあらわす構造体へのポインタ
 */
int setftime(int handle, struct ftime *ftimep)
{
    RAX = 0x5701;
    RBX = handle;
    RCX = ((unsigned *)ftimep)[0];
    RDX = ((unsigned *)ftimep)[1];
    intr(0x21, &regs);
    return RCY ? -1 : 0;
}

/*
 *  データの書き込み
 *      handle: ファイルハンドル
 *      buf: データを書き込むためのバッファ
 *      len: 書き込むバイト数
 *  戻り値
 *      実際に書き込まれたバイト数
 */
int _write(int handle, void *buf, unsigned len)
{
    RBX = handle;
    RCX = len;
    RDS = FP_SEG(buf);
    RDX = FP_OFF(buf);
    DOS(0x40);
    return RCY ? -1 : RAX;
}

/*
 *  多項式の計算
 *      x: x
 *      degree: 次数
 *      coeffs: 係数のリスト
 */
double poly(double x, int degree, double coeffs [])
{
    double y;

    for (y = 0.0; degree >= 0; degree--)
        y = x * y + coeffs[degree];
    return y;
}

#endif /* defined(__TURBOC__) */
```

特集 Cの学び方・教え方

Learning C

TURBO C++ for WINDOWS 3.0

統合環境を備えたWindows用超高速コンパイラ

怡土昌彦

BORLAND社から，また新しいC++コンパイラがリリースされた．TURBO C++ FOR WINDOWS 3.0である．BORLAND C++ 3.0もほぼ同時期にリリースされたが，ここではTURBO C++のほうに的を絞って紹介する．

C++は２本柱

今までBORLAND社の製品でWindowsに対応していたC++は，BORLAND C++（BC++）しかなかった．しかしBC++は高価すぎて手が出なかった向きがあったのではないだろうか．しかし，再び安価で高速なC++がTURBOの名称を掲げて還ってきたのである．しかも，Windowsに完全対応してだ．BORLANDの発表によると，BC++はプロフェッショナル向け，TURBO C++ FOR WINDOWS（TC++）はその他一般向けとしていくとのことである．ちょうどMicrosoft社がMicrosoft C（MS-C）とQuickC（QC）をリリースしているのと同じイメージになるのではないだろうか．

インストール

インストールプログラムは，Windowsアプリケーションになった．その結果，PC-9800シリーズなどでもインストールができるが，コンパイラなどは相変わらずマシン依存なので動作の保証はできない．また，すべてのメモリモデルやクラスライブラリなどをインストールすると，実に約20MB強のディスク容量が必要になる．これからパソコンの世界はハードディスクやメモリは湯水のように消費していく時代になるのではないだろうか．筆者は200MBのディスクを２台導入しているのだが，ほとんど一杯になりつつある．すべてがインストールされるとリスト１のようになり，プログラムマネージャには画面１のようなグループが登録される．実際にはWindowsディレクトリに転送されたDLLもいくつか存在している．

アップグレードの内容

今回のアップグレードで変更された点を簡単にあげてみよう．

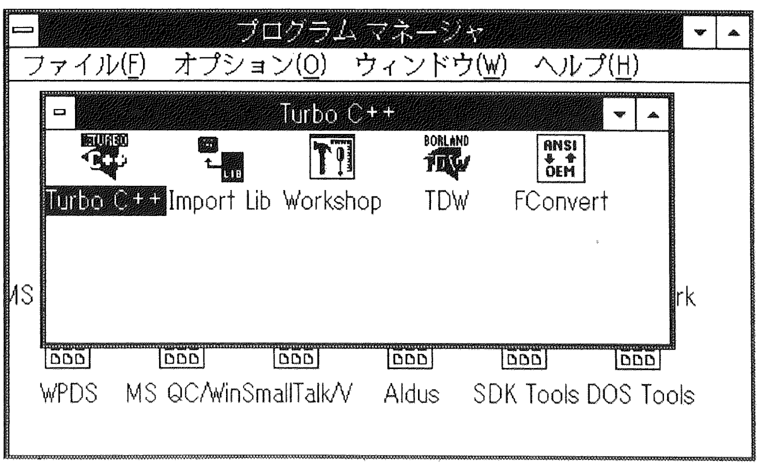
C++2.1のサポート
統合環境のWindows対応
SpeedBarのサポート
ObjectWindowsの標準装備
ObjectBrowserの実装
WindowsAppのサポート
リソース編集ツールの標準装備

統合環境のWindows対応

なんといっても今回の目玉は，統合環境がWindowsに完全対応したことであろう（画面２）．コーディングから実行までがすべてWindowsの上で行えるわけだ．ほとんど従来のDOS版統合環境と変わらない操作性を実現しているので，難なく新しい環境へ移行ができる．しかしすべてが誉められるわけではない．ステップ実行やブレークポイントの設定などのデバッグ機能が統合環境から削られてしまっているのだ．かわりに付属しているTurobo Debugger for Windows（TDW）を起動できるが，キャラクタモードでしか動作しない．つまりデバッグ中の実行画面（表示中のウィンドウ）はWindowsで見えるが，ソースはWindows上では見ることができないのである（画面３）．

またDOS/Vでは日本語モードでのデバッグができないのも痛い．Microsoft社のQuickC for Windows（QCW）はC++ではないが，デバッグも統合環境下で可能なことを考えると，やや統合環境に難ありと思える節

画面１



リスト1　インストール後のファイル

```
c:¥tcwin¥ :
bin
classlib
doc
examples
filelist.doc
include
lib
owl
readme
unpak.exe

c:¥tcwin¥bin¥ :
ati.dll
bc30rtl.dll
fconvert.exe
hc.exe
implibw.exe
loadbwcc.exe
noname00.exe
noname00.obj
owl.dll
tbrowser.dll
tcconfig.tc
tcdefw.dpr
tcdefw.dsk
tclass.dll
tcw.exe
tcwhelp.hlp
tddebug.386
tdstrip.exe
tdump.exe
tdw.exe
tdwhelp.tdh
tdwinst.exe
temc.exe
tseng.dll
windebug.dll
winstub.exe
worked1.dll
worked2.dll
worked3.dll
worked4.dll
worked5.dll
workhelp.hlp
worklib1.dll
worklib2.dll
workshop.exe

c:¥tcwin¥classlib¥ :
examples
include
lib
objs
source

c:¥tcwin¥classlib¥examples¥ :
directry.cpp
directry.h
directry.prj
filedata.cpp
filedata.h
lookup.cpp
lookup.prj
queuetst.cpp
queuetst.prj
reverse.cpp
reverse.prj
strngmax.cpp
strngmax.prj
testdir.cpp

c:¥tcwin¥classlib¥include¥ :
abstarry.h
array.h
arrays.h
assoc.h
bag.h
bags.h
btree.h
checks.h
clsdefs.h
clstypes.h
collect.h
contain.h
dbllist.h
deque.h
deques.h
dict.h
dlistimp.h
hashtbl.h
ldate.h
list.h
listimp.h
ltime.h
memmgr.h
object.h
priortyq.h
queue.h
queues.h
resource.h
set.h
sets.h
shddel.h
sortable.h
sortarry.h
stack.h
stacks.h
stdtempl.h
strng.h
timer.h
vectimp.h

c:¥tcwin¥classlib¥lib¥ :
tclasdbl.lib
tclasdbs.lib
tclasdll.lib
tclassl.lib
tclasss.lib

c:¥tcwin¥classlib¥objs¥ :
dl
dll
ds
l
s

c:¥tcwin¥classlib¥objs¥dl¥ :
bidsdbl.prj
tclasdbl.prj

c:¥tcwin¥classlib¥objs¥dll¥ :
bidsdll.prj
tclasdll.prj

c:¥tcwin¥classlib¥objs¥ds¥ :
bidsdbs.prj
tclasdbs.prj

c:¥tcwin¥classlib¥objs¥l¥ :
bidsl.prj
tclassl.prj

c:¥tcwin¥classlib¥objs¥s¥ :
bidss.prj
tclasss.prj

c:¥tcwin¥classlib¥source¥ :
abstarry.cpp
array.cpp
assoc.cpp
babstary.cpp
bdict.cpp
bids.def
bsortary.cpp
btree.cpp
btreeinn.cpp
btreelfn.cpp
clasmain.cpp
collect.cpp
contain.cpp
dbllist.cpp
deque.cpp
dict.cpp
hashtbl.cpp
ldate.cpp
list.cpp
ltime.cpp
memmgr.cpp
object.cpp
sortarry.cpp
stack.cpp
strng.cpp
tclass.def
timer.cpp
tmplinst.cpp

c:¥tcwin¥doc¥ :
ansi.doc
asmdebug.tdw
brief.tem
bwccapi.rw
bwccstyl.rw
cmacros.tem
custcntl.rw
dbgboard.tdw
defaults.tem
dosedit.tem
epsilon.tem
hdwdebug.tdw
helpme!.doc
helpme!.tdw
manual.rw
tdwinst.tdw
util.doc
utils.tdw
winmem.doc

c:¥tcwin¥examples¥ :
bitmap.cpp
bitmap.def
bitmap.dll
bitmap.dsk
bitmap.h
bitmap.obj
bitmap.prj
bitmap.rc
bitmap.res
bullet.bmp
chkboff.bmp
chkbon.bmp
codec.bmp
continue.bmp
ddeclnt.c
ddeclnt.h
ddeclnt.obj
ddesrvr.c
ddesrvr.h
dlldemo.cpp
dlldemo.def
dlldemo.ico
dlldemo.prj
dlldemo.rc
dlldemo.res
dlldemo1.bmp
dlldemo2.bmp
dllshell.cpp
dllshell.obj
done.bmp
ffind.c
ffind.def
ffind.dsk
ffind.h
ffind.ico
ffind.prj
ffind.rc
ffind.res
hdump.c
hdump.def
hdump.dsk
hdump.h
hdump.ico
hdump.prj
hdump.rc
hdump.res
help.cur
helpex.c
helpex.def
helpex.dlg
helpex.h
helpex.hlp
helpex.hpj
helpex.ph
helpex.prj
helpex.rc
helpex.res
helpex.rtf
helpicon.bmp
helpids.h
keys.rtf
max2icon.bmp
mouse.bmp
msgproc.c
msgproc.h
msgproc.obj
msgwnd.c
msgwnd.def
msgwnd.dsk
msgwnd.h
msgwnd.ico
msgwnd.prj
msgwnd.rc
msgwnd.res
myfuncs.cpp
myfuncs.h
mymain.cpp
myprog.dsk
td
terms.rtf
todo.def
todo.dsk
todo.exe
todo.prj
todo.sym
tododefs.h
tododlgs.cpp
tododlgs.h
tododlgs.obj
todolist.cpp
todolist.h
todolist.obj
todowin.cpp
todowin.h
todowin.obj
todowin.rc
todowin.res
tstapp.c
tstapp.def
tstapp.dsk
tstapp.exe
tstapp.h
tstapp.ico
tstapp.obj
tstapp.prj
tstapp.rc
tstapp.res
whello.cpp
whello.def
whello.ico
whello.mak
whello.prj
whello.rc
whello.res
winword.bmp
workshop
x.dsk
x.prj

c:¥tcwin¥examples¥td¥ :
tddemo.c
tddemo.exe
tddemo.prj
tddemob.c
tddemob.exe
tddemob.prj
tdodemo.cpp
tdodemo.def
tdodemo.exe
tdodemo.h
tdodemo.ico
tdodemo.prj
tdodemo.rc
tdodemo.res
tdodemob.cpp
tdodemob.exe
tdodemob.h
tdodemob.prj

c:¥tcwin¥examples¥workshop¥ :
bitbnapp.c
bitbnapp.res
bitbtn.c
bitbtn.def
bitbtn.prj
bitbtn.res
bitbtnco.h
btnbnapp.prj
rwcdemo.cpp
rwcdemo.def
rwcdemo.h
rwcdemo.prj
rwcdemo.rc
rwcdemo.res
rwcdemob.prj
rwcdemob.rc
rwcdemob.res
rwcdemoc.h
rwcdlg.cpp
rwcdlg.h
rwcwnd.cpp
rwcwnd.h
rwdemo.cur
rwdemo.ico
statbar.bmp
statline.bmp

c:¥tcwin¥include¥ :
alloc.h
assert.h
bcd.h
bios.h
bwcc.h
complex.h
conio.h
ctype.h
custcntl.h
dde.h
dir.h
direct.h
dirent.h
dos.h
drivinit.h
errno.h
fcntl.h
float.h
fstream.h
generic.h
io.h
iomanip.h
iostream.h
limits.h
locale.h
locking.h
malloc.h
math.h
mem.h
memory.h
new.h
process.h
search.h
setjmp.h
share.h
signal.h
stdarg.h
stddef.h
stdio.h
stdiostr.h
stdlib.h
string.h
strstrea.h
sys
time.h
utime.h
values.h
varargs.h
windows.h
windows.inc
_defs.h
_null.h
c:¥tcwin¥include¥sys¥ :
stat.h
timeb.h
types.h

c:¥tcwin¥lib¥ :
bwcc.lib
c0dc.obj
c0dl.obj
c0dm.obj
c0ds.obj
c0wc.obj
c0wl.obj
```


Cプログラミング用語
予約語　識別子として使えないキーワード．ANSI Cでは32語が予約語になっている．

```
c0wm.obj
c0ws.obj
crtldll.lib
cwc.lib
cwl.lib
cwm.lib
cws.lib
import.lib
mathwc.lib
mathwl.lib
mathwm.lib
mathws.lib
wildargs.obj

c:¥tcwin¥owl¥ :
doc
examples
include
lib

c:¥tcwin¥owl¥doc¥ :
bwcc-owl.doc
helpme!.doc
owl.doc
owldll.doc

c:¥tcwin¥owl¥examples¥ :
bonk
bscrlapp.cpp
bscrlapp.prj
bscrlapp.rc
bscrlapp.res
btntest.cpp
btntest.prj
calc
calldll.cpp
calldll.h
calldll.prj
calldll.rc
calldll.res
cboxtest.cpp
cboxtest.prj
cbxttest.cpp
cbxttest.prj
cctltest.cpp
cctltest.h
cctltest.prj
cctltest.rc
cctltest.res
cdlgdll.cpp
cdlgdll.def
cdlgdll.h
cdlgdll.prj
cdlgdll.rc
cdlgdll.res
checkers
colordlg.cpp
colordlg.dlg
colordlg.h
colordlg.rch
cursapp.cpp
cursapp.prj
cursor
dctltest.cpp
dctltest.h
dctltest.prj
dctltest.rc
dctltest.res
dialtest.cpp
dialtest.h
dialtest.prj
dialtest.rc
dialtest.res
dllhello.cpp
dllhello.def
dllhello.h
dllhello.prj
editapp.cpp
editapp.prj
editapp.rc
editapp.res
edittest.cpp
edittest.prj
ewndtest.cpp
ewndtest.h
ewndtest.prj
ewndtest.rc
ewndtest.res
fileapp
gboxtest.cpp
gboxtest.prj
gdidemo
helloapp.cpp
helloapp.prj
insttest.cpp
insttest.prj
lboxtest.cpp
lboxtest.prj
lbxttest.cpp
lbxttest.prj
mdiapp.cpp
mdiapp.prj
mdiapp.rc
mdiapp.res
mditest.cpp
mditest.h
mditest.prj
mditest.rc
mditest.res
mfileapp
mlbxttst.cpp
mlbxttst.prj
notitest.cpp
notitest.h
notitest.prj
notitest.rc
notitest.res
paint
paltest.cpp
paltest.dlg
paltest.h
paltest.prj
paltest.rc
paltest.res
popup.cpp
popup.prj
popup.rc
popup.res
progtalk
sbartest.cpp
sbartest.prj
scribapp.cpp
scribapp.prj
scrolapp.cpp
scrolapp.prj
stattest.cpp
stattest.prj
steps
sysinfo
trantest.cpp
trantest.h
trantest.prj
trantest.rc
trantest.res
ttt
usecdll.cpp
usecdll.h
usecdll.prj
usecdll.rc
usecdll.res
usecdll2.cpp
usecdll2.h
usecdll2.prj
usecdll2.rc
usecdll2.res
vdlgapp.cpp
vdlgapp.prj
vdlgapp.rc
vdlgapp.res

c:¥tcwin¥owl¥examples¥bonk¥ :
board.bmp
board2.bmp
bonk.cpp
bonk.h
bonk.prj
bonk.rc
bonk.res
bonk2.rc
bonk2.res
critter.ico
critter2.ico
dead.bmp
dead2.bmp
gameove2.bmp
gameover.bmp
live.bmp
live2.bmp
malet.cur
malet2.cur
maletdo2.cur
maletdow.cur

c:¥tcwin¥owl¥examples¥calc¥ :
calc.cpp
calc.ico
calc.prj
calc.rc
calc.res

c:¥tcwin¥owl¥examples¥checkers¥ :
blackkng.bmp
blackman.bmp
board.cpp
board.h
checkers.cpp
checkers.h
checkers.ico
checkers.prj
checkers.rc
checkers.res
hand.cur
handpc.cur
info.cpp
info.h
redking.bmp
redman.bmp

c:¥tcwin¥owl¥examples¥cursor¥ :
cursor.cpp
cursor.def
cursor.h
cursor.ico
cursor.prj
cursor.rc
cursor.res

c:¥tcwin¥owl¥examples¥fileapp¥ :
fileapp.cpp
fileapp.prj
fileapp.rc
fileapp.res

c:¥tcwin¥owl¥examples¥gdidemo¥ :
arty.cpp
arty.h
artypriv.h
backgrnd.bmp
bitblt.cpp
bitblt.h
demobase.h
font.cpp
font.h
gdidemo.cpp
gdidemo.prj
gdidemo.rc
gdidemo.res
line.cpp
line.h
monoship.bmp
ship.bmp

c:¥tcwin¥owl¥examples¥mfileapp¥ :
mfileapp.cpp
mfileapp.h
mfileapp.prj
mfileapp.rc
mfileapp.res

c:¥tcwin¥owl¥examples¥paint¥ :
ellipse.cur
ellipse.ico
fellipse.cur
fellipse.ico
fill.cur
fill.ico
frect.cur
frect.ico
owlpaint.ico
paint.cpp
paint.prj
paint.rc
paint.res
pen.cur
pen.ico
rect.cur
rect.ico
c:¥tcwin¥owl¥examples¥progtalk¥ :
ddeclien.cpp
ddeclien.h
progtalk.cpp
progtalk.h
progtalk.prj
progtalk.rc
progtalk.res

c:¥tcwin¥owl¥examples¥steps¥ :
helpwind.cpp
helpwind.def
helpwind.h
step1.cpp
step1.prj
step10.cpp
step10.prj
step2.cpp
step2.prj
step3.cpp
step3.prj
step4.cpp
step4.prj
step5.cpp
step5.prj
step6.cpp
step6.prj
step7.cpp
step7.prj
step8.cpp
step8.prj
step9.cpp
step9.prj
steps.def
steps.h
steps.rc
steps.res

c:¥tcwin¥owl¥examples¥sysinfo¥ :
sysinfo.cpp
sysinfo.h
sysinfo.prj
sysinfo.rc
sysinfo.res

c:¥tcwin¥owl¥examples¥ttt¥ :
ttt.cpp
ttt.h
ttt.ico
ttt.prj
ttt.rc
ttt.res

c:¥tcwin¥owl¥include¥ :
appdict.h
applicat.h
bbutton.h
bchkbox.h
bdivider.h
bgrpbox.h
bradio.h
bstatbmp.h
bstatic.h
button.h
bwindow.h
checkbox.h
combobox.h
control.h
dialog.h
edit.h
editacc.rc
editmenu.rc
editwnd.h
fileacc.rc
filedial.dlg
filedial.h
filemenu.rc
filewnd.h
groupbox.h
inputdia.dlg
inputdia.h
listbox.h
mdi.h
module.h
objstrm.h
owl.h
owldefs.h
owlrc.h
radiobut.h
safepool.h
scrollba.h
scroller.h
static.h
stdwnds.dlg
tcollect.h
windobj.h
window.h

c:¥tcwin¥owl¥lib¥ :
owl.def
owl.lib
owldll.def
owlwc.lib
owlwl.lib
owlwm.lib
owlws.lib
```

がある．ただし，TDWを用いたほうが高機能なデバッグを行えることも事実である．

SpeedBarのサポート

しかし，統合環境の機能が削られているだけではなく強化も行われている．SpeedBarと呼ばれる機能だ．ひんぱんに使われる機能をアイコンボタン化してあり，いちいちメニューからチョイスしなくても簡単に利用することができる．ちょうどペイントブラシの左端に機能別のパレットが並んでいるような機能だと考えてもらいたい（画面4）．

ユーザーが，任意にボタンをカスタマイズすることはできないが，状況に応じてボタンのバリエーションが変化する．この機能はアイコンのイメージと機能が結びつくまでに慣れが必要だが，使い慣れてしまうとたいへん便利な機能である．ユーザーに任意のカスタマイズができれば，なお一層便利な機能になるのではないだろうか．

ObjectWindowsの標準装備

TC++には，Windows用クラスライブラリが付属している（図1）．今までBC++を単独で利用していたユーザーにとっては，もっと早くこれが付属していて欲しかったと思う代物だ．何もない状態からクラスライブラリを作成するにはたいへんな労力を必要とする．そもそもオブジェクト思考プログラミングとは，各オブジェクトの“ふるまい”をメソッド単位にクラスへ記述し，それらを組み合わせてプログラミングすることをいう．もし，実用となるクラスライブラリが存在するのならば，それらを組み合わせるだけでプログラミングが完了するために労力は半分以下で済む．

ただ1つ残念なのは，コンパイラはC++ 2.1をサポートしているのに，クラスライブラリは2.0のままな点だ．C++ 2.0は図1を見ればわかるとおり，複数の親を持つクラスがまだ許されているのである．筆者のようにSmalltalkなどに慣れているユーザーには，“Object”以外に同列のクラスが存在するのは奇妙な気分である．

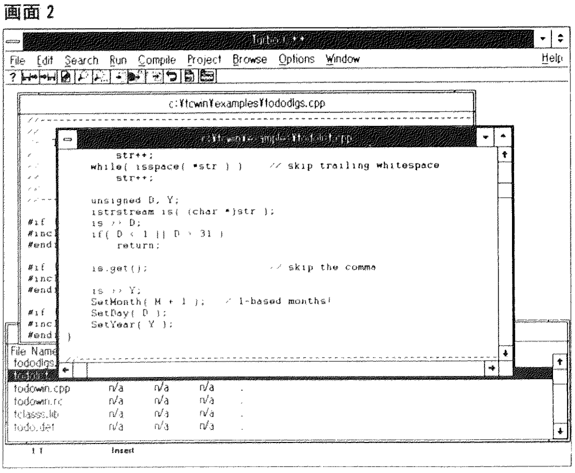
ObjectBrowserの実装

ObjectBrowserはOOP環境においては必須機能だといえる．この機能を使って各オブジェクトの継承（階層）のTREE表示（画面5），メソッドの一覧表示（画面6），目的クラスへの移動を行う．従来この機能は一部分だけTDWで利用できたのだが，統合環境ではより強化されて利用できるので一層便利になった．しかしブラウザの情報はプログラムに格納されているので，コンパイルエラーなどがおきている間はブラウザの恩恵にあずかることができない．この点は今後の機能向上に期待したい．

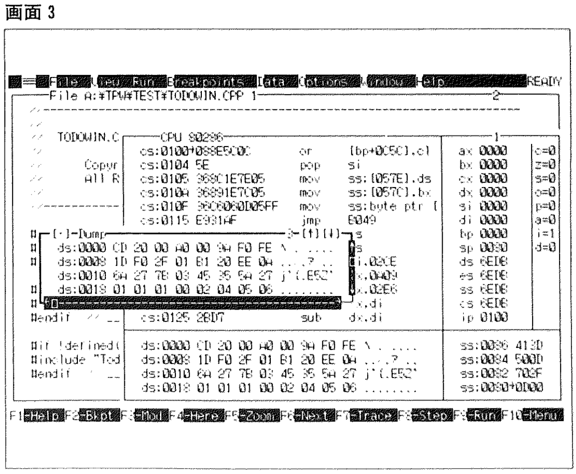
WindowsAppのサポート

作成したプログラムは，即Windows環境下で動作させることができる．しかも，DLLの生成も可能となっている（画面7）．その反面Turbo Pascal for Windows同様DOSプログラムの作成はできなくなってしまった．次のバージョンでは，ぜひDOSプ

画面2



画面3



Cプログラミング用語
リダイレクト　標準入力や標準出力先を変更すること．普通は標準入力はキーボード，標準出力はディスプレイが指定されている．

ログラムも作成できるようにしてもらいたい.

リソース編集ツールの標準装備

日本語版のBC++ではおなじみだが, 英語版のBC++には付属していなかったResource WorkShop (RWS) のニューバージョンが付属している. アバウト画面の絵も, 男性から女性になっている(画面8).

ヘルプファイルを作成するツールは, 従来どおりHelp Compilerである. こんどはRWSのようなヘルプ構築ツールを作って欲しいものだ.

マニュアル

「TURBO C++ USER'S GUIDE」「TURBO C++ PROGRAMMER'S GUIDE」「OBJECT WINDOWS USER'S GUIDE」「TURBO DEBUGGER USER'S GUIDE」「RESOURCE WORKSHOP USER'S GUIDE」の5冊で構成されている. しかし, Cのランタイムライブラリリファレンスや Windowsの細かいAPIなどはヘルプとして提供されている(画面9). もし紙に記述されたマニュアルが欲しければ, 必要に応じて別途書店などで類似するものを買い求める必要があるだろう.

Learning C 最後に

Windowsアプリケーションは, マシン依存しないことが鉄則なのだが, このコンパイラで作成したアプリケーションはスタートアップルーチンがマシン依存しているためにIBMとその互換機に依存する. この点は前々から気になっているのだが, ボーランド(株)は本社にスタートアップルーチンなどの基本ライブラリがマシン依存していることを注意していないのではないだろうか. また米国ではまもなくWindows 3.1が発表されるが, 現在のTC++にはそれらのAPIは含まれていない. ボーランドは3.1発表後, 1か月以内に対応版を発表するといっている. できるならば, アップデート

画面4

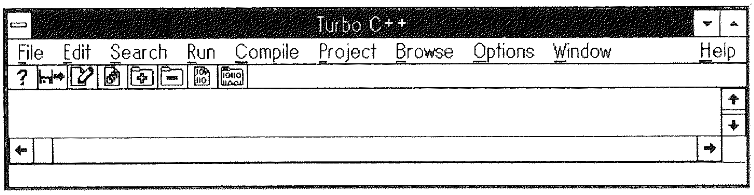
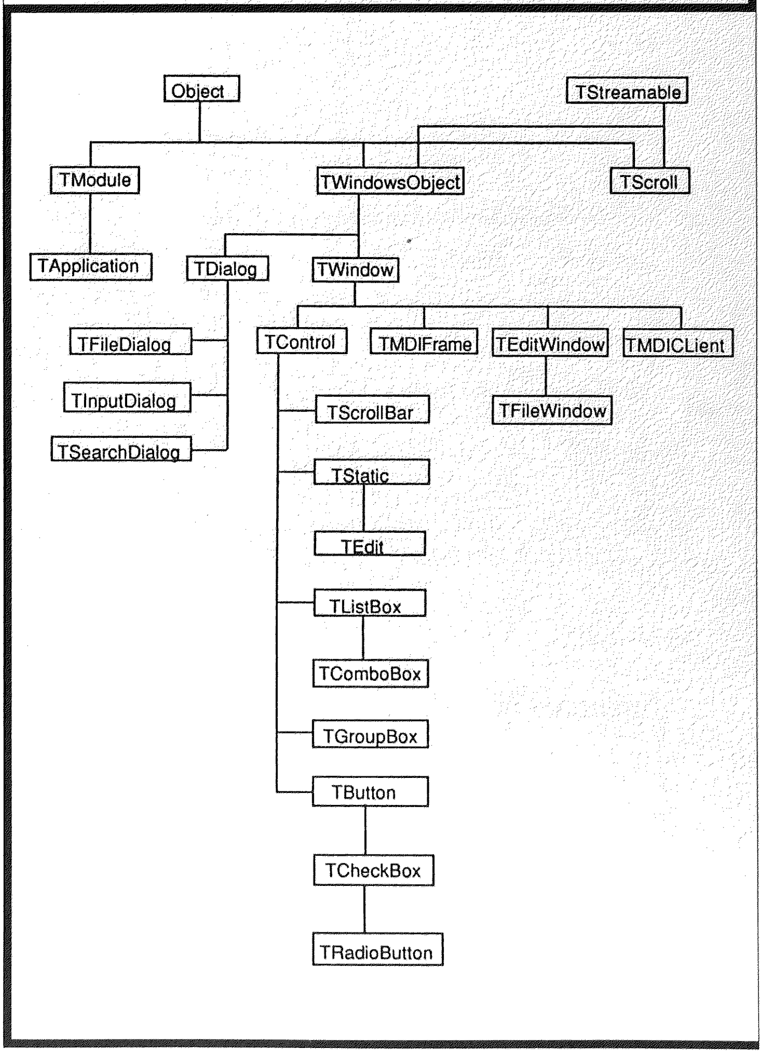


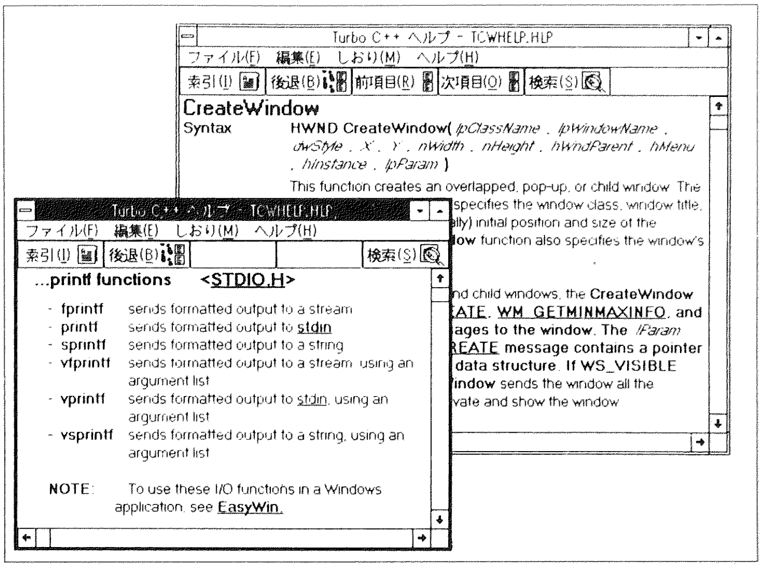
図1 ObjectWindowsの階層構造



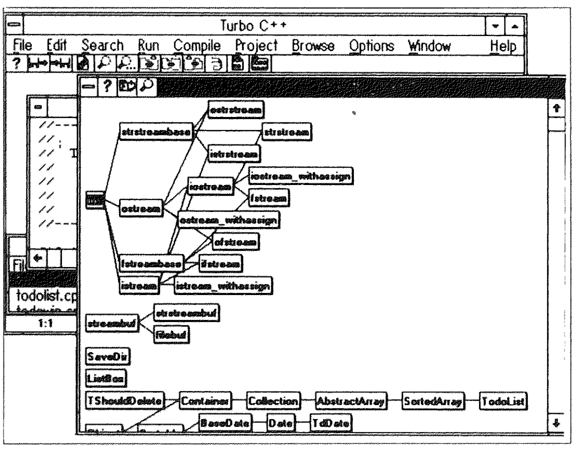
価格をより抑えた設定にしてもらいたいものだ．(編集部注：日本語版 TCW で作成したアプリケーションについては機種依存するようなことはない．)

いろいろと文句をいってきたが，BC＋＋でやや遅くなったコンパイルスピードが再び復活している．すごい勢いでコンパイルが進むのは見ていても爽快なくらいだ．リンカも申し分ないスピードといえる．IBM 御用達の言語メーカーになったのだから，今後はライブラリだけでも機種依存をしないように修正されていくことに期待したい．

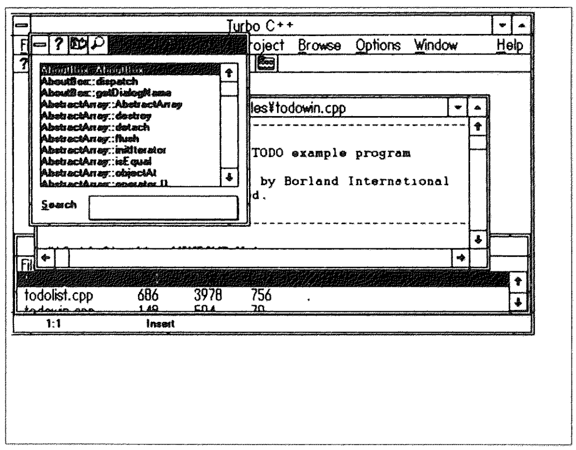
画面 9



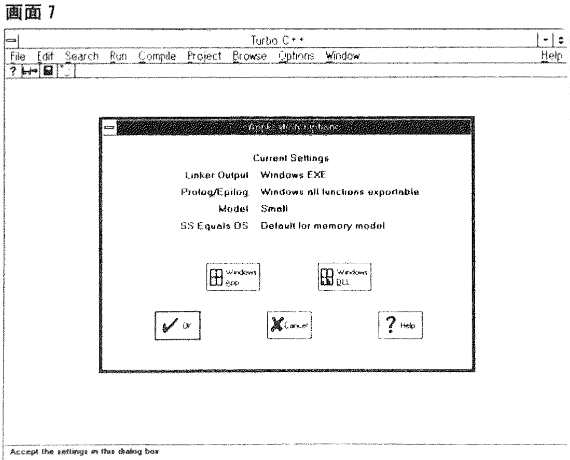
画面 5



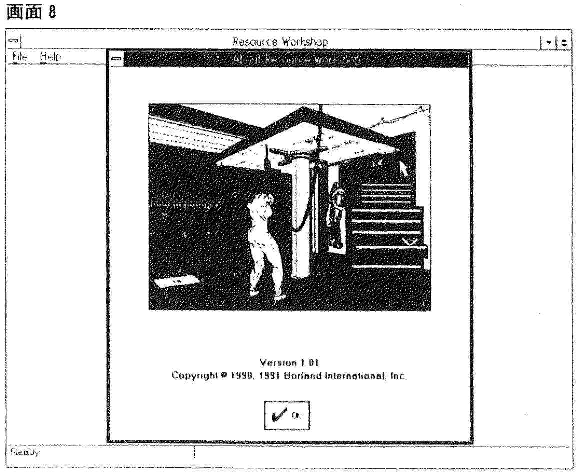
画面 6



画面 7

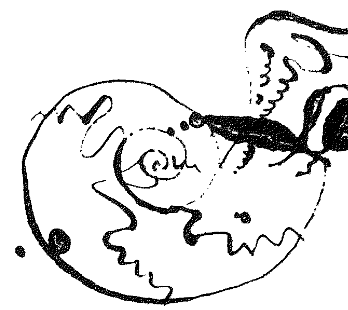


画面 8



C プログラミング用語
論理演算子　演算結果が記号論理学の規則に従い，オペランドのとりうる値からのみなる演算．

Illustration : Masami Nozue



アルゴリズム篇 第5回

ツリー・その1

実践C塾

三代目塾頭
㈱ジャステックラボ
加野島英渡／山本博昭

前回までデータ検索のためのアルゴリズムとして,「シリアルサーチ」,「バイナリサーチ」の2つをみてきたが，今回はデータの大小関係を表現する構造手法の1つである「ツリー」を取り上げる.

先月までのあらすじ

データ群の中から目的のデータを検索するためのアルゴリズムとして，先頭から順に検索していく「シリアルサーチ」，ソート済みのデータ群を高速に検索する「バイナリサーチ」という2つの手法を2回にわたって考察してきた．バイナリサーチはシリアルサーチと比べてはるかに高速であるが，検索対象となるデータ群がソート済みであることという制約があった．ただし，データが何もない状態から処理をはじめるのであれば，あるデータが既存のものかどうかを調べるためにバイナリサーチを行い，見つからなかった時には最後に比較したデータ位置の右または左(最終比較結果によって決まる)に，そのデータを新規追加していくという手法がある．この手法を用いると，データ群は結果的につねにソートされている状態で更新されていくことになる．先月はこのアルゴリズムを応用して，テーマツールの改造を行った．改造後のテーマツールでは，全ファイルを読み終えて，「単語リスト」が完成した時点で，その単語リストはすでにコード順に整列されている状態になっているため，表示時のソートが不要になった．

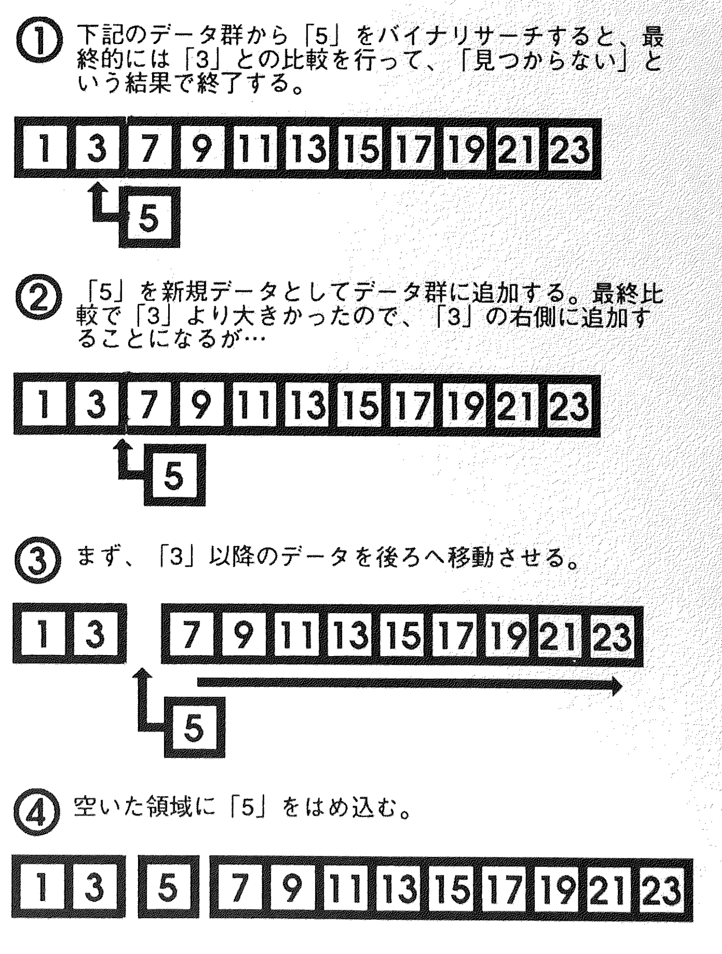
今月の重点項目

まず，先月の解説で誌面が足りなかったためにふれることのできなかったバイナリサーチの問題点を1つ補足しておこう．問題点を明確にするために，特定の単語をバイナリサーチで検索し，見つからなかったときはその時点でポインタがさしている箇所に新規データとして登録するという処理をもう一度確認してみよう．

図1でわかるように，新規データを追加するときに，追加箇所以降のデータを移動させている．データ数が少ない場合はそれほどコストはかからないだろうが，巨大なデータ群の先頭付近に新規データを追加することを考えると気が重くなってくる．

そこで今月は，このようなデータの移動を行わなくとも，バイナリサーチと同等の効率のよい検索ができるような手法を考えてみることにする．

図1　バイナリサーチでのデータの新規登録



ツリー

先月号で紹介した，バイナリサーチをしながらデータを構築していくという手法も，結果的には直線状に並んだデータ群を処理しているわけで，データ群の途中に新規データを追加しようとすると，先ほど書いたようなデータの移動が発生してしまう．これを嫌うならば，データの構造自体から見直しをするべきである．というわけで，今月はランダムに出現してくるデータを効率よく構築するためのデータ構造を考えていくことにしよう．

データの大小関係を表現する構造手法の1つとして，「ツリー」があげられる．「ツリー」とはその名が示すとおり，「木」のことである．あるデータを始まりとして，そのデータと次のデータの関係(大小関係など)を接続し，そのまた次のデータというように枝分かれ状にデータ群を構築していくのが「ツリー」の基本的な考え方である．具体的なイメージとして，図2のような家系図を思い浮かべていただきたい．

基準となる祖父太郎・祖母江夫婦というデータから，その子供，その子供の子供というようにデータ群が構築されている（横一列のデータは，右のほうにいくほど年長という取り決めにしておく）．このような表現になっていれば，誰の子供が誰であるか，また誰は誰と誰の親であるかなどが一目でわかるようになる．また，直接の関係だけではなく，2代前をたどることも容易になる．

ツリー構造は，「木」を意味しているので，使われる用語もそれになぞらえている．図2で，データの始まりとなっている祖父太郎・祖母江夫婦を「根(ルート)」と呼ぶ．そして彼らの子供たちは「節（ノード)」と呼ぶことになる．ただし，子供がまだいない者（自分から下に線が出ていない者）は「葉（リーフ)」と呼ばれる．またツリーでは，あるデータの下につながっているデータのことを「子」，上につながっているデータを「親」と呼ぶ．「葉」は「子」をもたないデータであり，1つのツリー構造において，「親」のないデータは「根」のデータただ1つということになる．根が上になっていて，葉がいちばん下にあるのだからずいぶんと妙な木ではあるが，何がおこるかわからないコンピュータ業界のこととしてご勘弁いただきたい．

ツリーの作り方

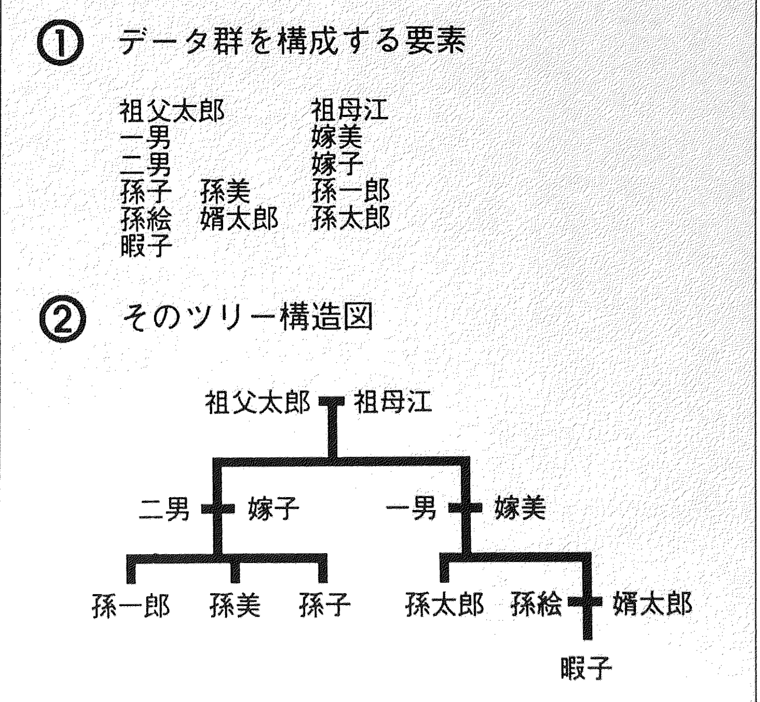
さて，話をもう少しプログラミングに近づけていこう．家族構成をツリー表現にすると，1つの「節」から出てくるデータ(この場合は子供の数)は決まっていない．1つのデータがもつ「子」の数が一定していないわけである．これをそれぞれのデータの大小関係のみにしぼると，あるデータのもつ「子」は「自分より小さいもの」，「自分より大きいもの」の2つだけとなる．このような構造のツリー構造は「2分木」と呼ばれている．2分木の構築は簡単である．最初のデータが「根」となり，次にデータが届いた時に「根」と比較して，小さければ左側に，大きければ右側に「節」を作っていく．図3を見ながら具体的な数値を使って2分木構造の作り方をたどっていってみよう．

ツリーへのデータ追加

いま書いてきたデータ構築の手順が，そのままデータを追加する手順となる．もう一度，データを追加する際の手順をまとめてみると，

①追加しようとするデータを「根」と比較する．

図2　家族をツリー構造で表現する



②結果が小さければ左側，大きければ右側に子がないかどうか確認する．

③子がなければそこに新しい節を作る．

④子があればその子を追加データと比較する．

⑤②に戻って新しい節を作ることができるようになるまで繰り返す．

この手順でデータを追加すると，新しく追加するデータは必ずツリーにとっての「葉」となる．ツリーにデータを追加する際には，枝の途中にデータを挿入するということはないわけだ．バイナリサーチをしながらデータを追加していった先月のアルゴリズムと基本的に違っているのがここの部分である．図4は，さきほどのツリーに新規データを追加する際の手順をあらわしたものである．

図4　ツリーにデータを追加する手順

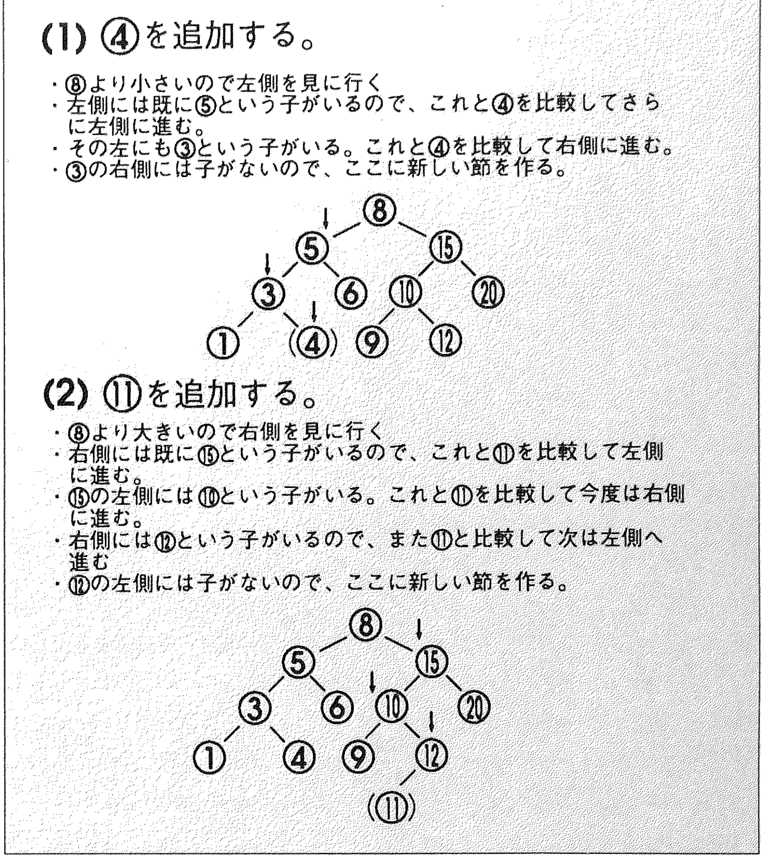
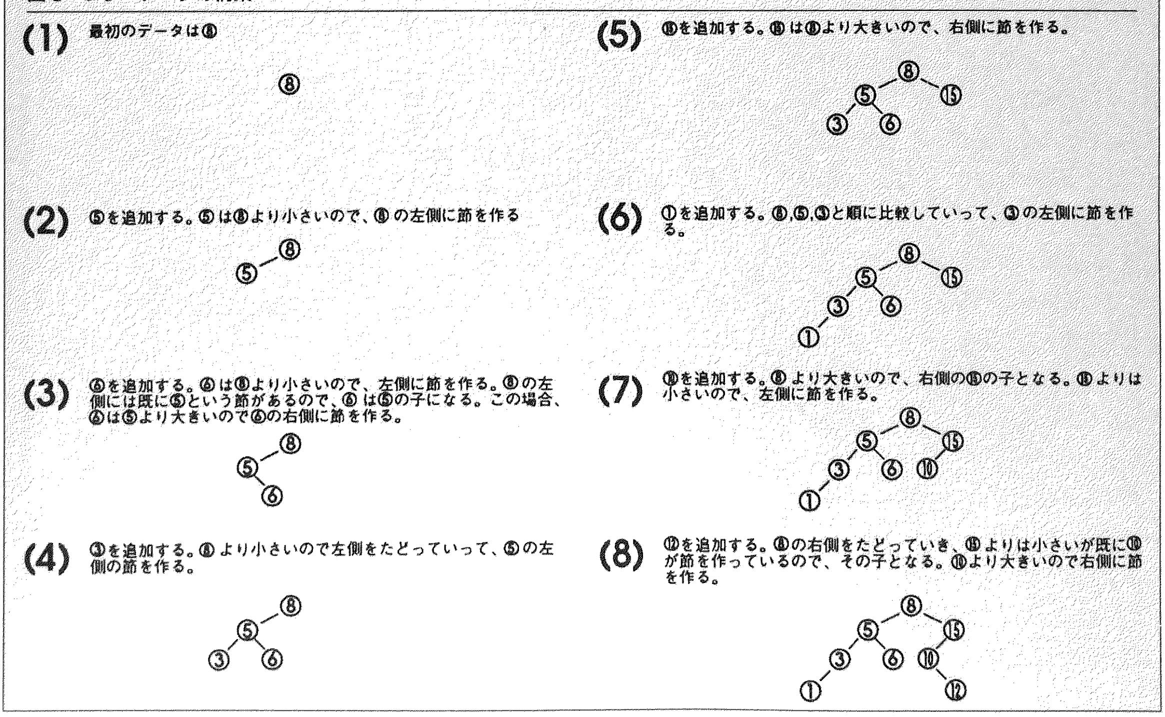


図3　ツリーデータの構築



ツリーデータの削除

次に，ツリー構造から特定のデータを削除する手順を考えてみる．ツリー構造は，親から子，そしてまたその子というようにデータが関連づけられているので，途中のデータをいきなり削除することができない．木の太い枝を根元から切ってしまうと，その先の枝やら葉やらもいっしょになって落ちてしまうからである．ツリーデータの削除は，対象となるデータがツリーの中のどこに位置しているかによって削除の手順が異なってくる．それぞれの場合に分けてみていくことにする．具体的な例は，図5の削除手順で確認していただきたい．

ツリーデータの表示

最後に，ツリーデータを表示する手順をまとめる．当然のことながらツリーデータは「根」から順に表示していっては意味がない．ここでは，「小さいものから大きいものへ」「大きいものから小さなものへ」という2つの手順をみていく．

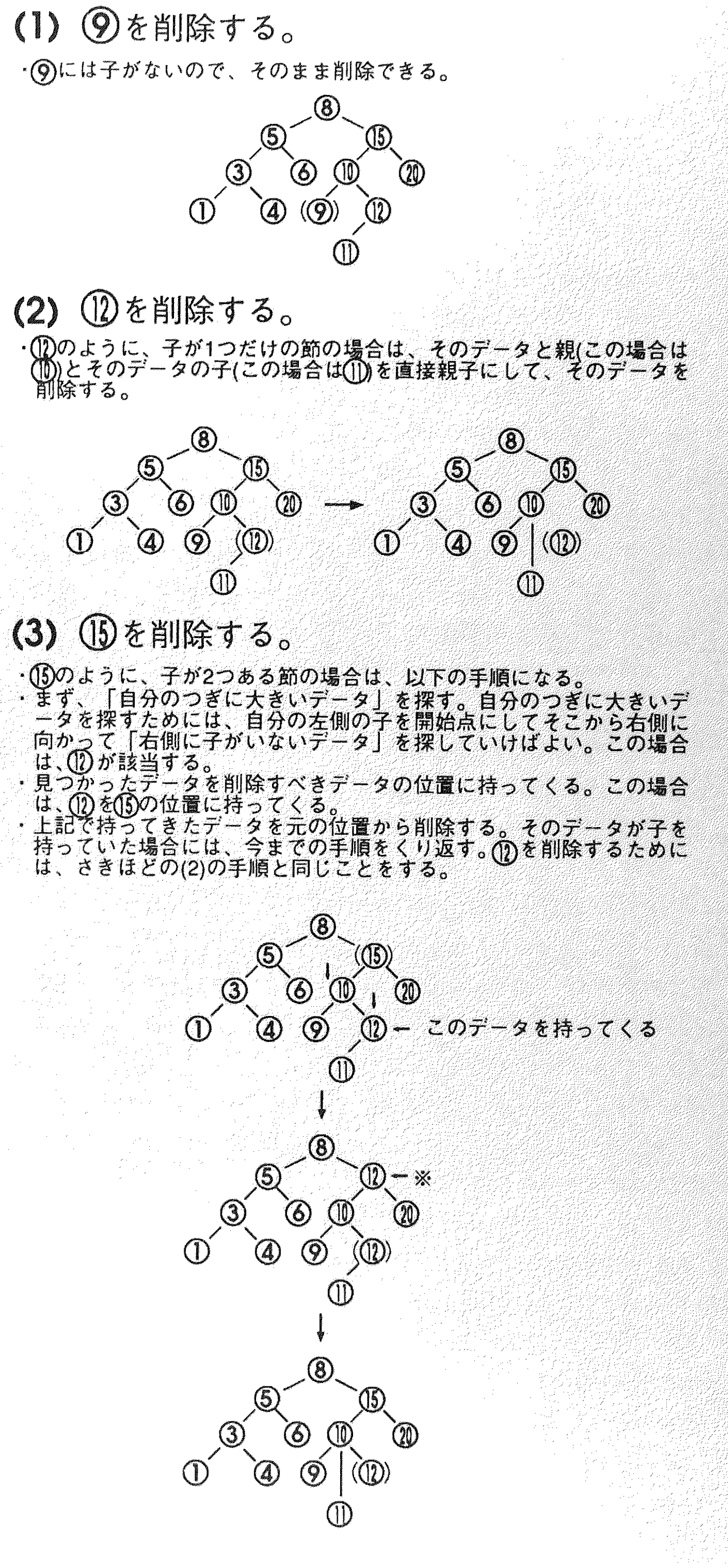
【小さなものから大きなものへ順に表示する】

この場合に考えることは，「データの中で最小のものを探す」ことと「次に小さなものを探す」方法である．この2つがわかれば，後はデータがなくなるまで同じことを繰り返していけばよい．

①最小のデータを探す

「根」のデータの左側の子からはじめて，その子がさらに左側の子をもっているかどうかをたどっていく．左側に子のないデータにいきあたったら，そのデータがツリーの中で最小のデータである．

図5　ツリーからデータを削除する手順



②次に小さなデータを探す

最小のデータが見つかったら，その次に小さなデータは以下の手順で探す

- 最小データの右に子がある場合
  右側の子が「葉」の場合は，そのデータ自身が次に小さい
  さらに子がある場合は，そのデータからスタートして①の要領でその中での最小データを探す
- 右に子がない場合は，最小データの親が次に小さなデータである．
  親がない（最小データが「根」だった）場合は，そこで処理は終り．

簡単にまとめると，こういう具合になる．

- 最小データの検索とは，「葉」になるまで左側の子をたどっていくことである．
- 次のデータは，そのデータが右に子をもっている場合はその子を基点にして，最小データの検索を行う．
- 右に子がない場合は，親が次のデータとなる．親もない場合はそこで処理は終了．

では，この手順を図6で具体的な数値を使って検証していくことにする．

【大きなものから小さなものへ順に表示する】

さきほどの場合とまったく逆に，まず「データの中で最大」のものを探す．

①最大のデータを探す

「根」のデータの右側の子からはじめて，その子がさらに右側の子をもっているかどうかをたどっていく．右側に子のないデータに行きあたったら，そのデータがツリーの中で最大のデー

図6　ツリーのデータを表示する手順（小から大へ）

(1) 根(⑧)から最小のデータを検索する。左側の子をたどっていって、「葉」であるデータを探していくと、①に行きつくのでこれを表示する。

(2) ①の右には子がないので、親にあがって③を表示する。

(3) ③の右に子がある。［④から右の処理］
　(3-1)④には子がないのでこれを表示する。

(4) ③の右の処理が終わったので、親にあがって⑤を表示する。

(5) ⑤の右に子がある。［⑥から右の処理］
　(5-1)⑥には子がないのでこれを表示する。

(6) ⑤の右の処理が終わったので、親にあがって⑧を表示する。

(7) ⑧の右に子がある。［⑮から右の処理］
　(5-1)⑮の左側に子があるので、左側にたどっていく
　(7-2)⑨が見つかったので、これを表示する。
　(7-3)⑨には子がないので、親にあがって⑩を表示する。
　(7-4)⑩の右側に子がある。
　　(7-4-1)⑫の左側に子があるので、左側にたどっていく
　　(7-4-2)⑪が見つかったので、これを表示する。
　　(7-4-3)⑪には子がないので親にあがって⑫を表示する。
　(7-5)⑩の右の処理が終わったので、親にあがって⑮を表示する。
　(7-6)⑮の右側に子がある。
　　(7-6-1)⑳には子がないのでこれを表示する。

(8) ⑧には親がない(根のデータである)ので、これで処理は終わり

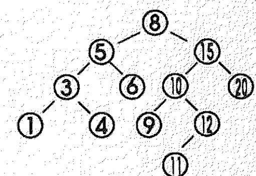


図7　ツリーのデータを表示する手順（大から小へ）

(1) 根(⑧)から最大のデータを検索する。左側の子をたどっていって、子がないデータを探していくと⑳が見つかるのでこれを表示する。

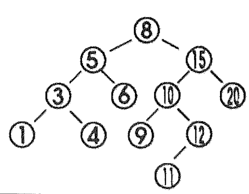
(2) ⑳には子がないので、親にあがって⑮を表示する。

(3) ⑮の左に子がある。［⑩から左の処理］
　(3-1)⑩から右にたどっていって⑫が見つかるので、これを表示する
　(3-2)⑫の左に子がある。［⑪から左の処理］
　　(3-2-1)⑪には子がないのでこれを表示する。
　(3-3)⑫の左の処理が終わったので、親にあがって⑩を表示する。
　(3-4)⑩の左に子がある。［⑨から左の処理］
　　(3-4-1)⑨には子がないのでこれを表示する。

(4) ⑮の左の処理が終わったので、親にあがって⑧を表示する。

(5) ⑧の左に子がある。［⑤から左の処理］
　(5-1)⑤から右にたどっていって⑥が見つかるので、これを表示する
　(5-2)⑥には子がないので、親にあがって⑤を表示する
　(5-3)⑤の左に子がある。［③から左の処理］
　　(5-3-1) ③から右にたどっていって④が見つかるので、これを表示する。
　　(5-3-2)④には子がないので、親にあがって③を表示する。
　　(5-3-3)③の左に子がある［①から左の処理］
　　　(5-3-3-1)①には子がないのでこれを表示する。

(6) ⑧には親がないので、処理はここで終わり。



タである.

②次に大きなデータを探す

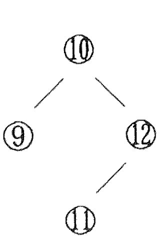
最大のデータが見つかったら,その次に大きなデータは以下の手順で探す.

- ・そのデータの左に子がある場合
  左側の子が「葉」の場合は，そのデータが次に大きい.
  左側の子にさらに子がある場合は，そのデータからスタートして①の手順でその中での最大データを探す.
- ・左に子がない場合は，いまのデータの親が次に大きなデータである.
  親がない（該当のデータが「根」）場合は，そこで処理は終り.

では，この手順を図7で具体的な数値を使って検証していくことにする.

ツリーデータのプログラミング

ツリー構造を，データの構築・追加・削除・表示と見てきた. では実際にこの構造をプログラムで表現するには，どういったコーディングを行えばよいのだろうか.



このような絵を書いてみたところで，これがプログラムになるとは思えない. この絵を実際のデータとしてコーディングするためには，各々のデータとそのデータの右側や左側にある子データとの関係を何らかの規則で表現することが必要になってくる. 1つのデータをC言語でいう「構造体」であらわすとすると，そのデータが右や左に子をもつことを表現するためには，その子データ構造体のポインタをもっていればよい. たとえば，上のツリーの絵は図8のようなデータ構造で表現できる. ここのデータ構造に必要な情報は以下のものである.

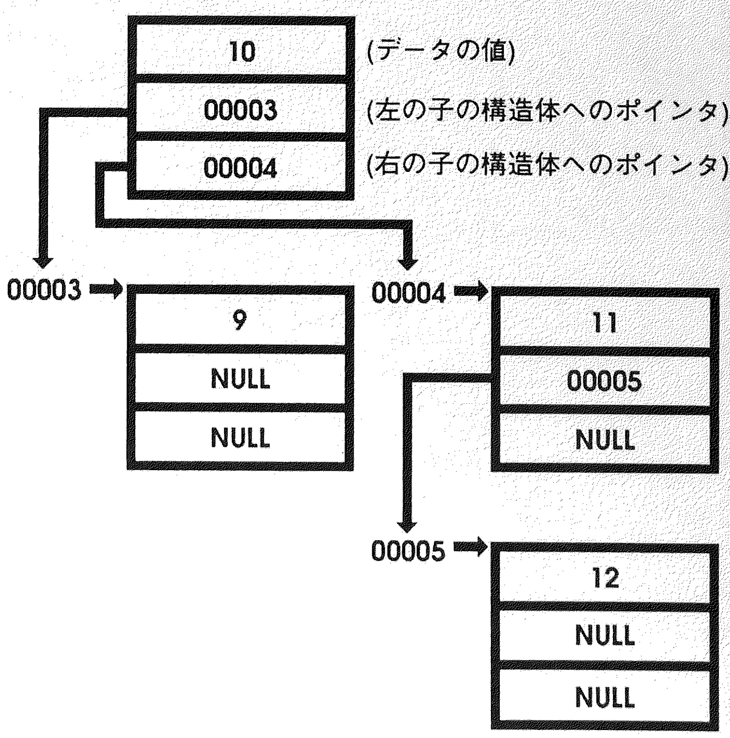
- **・データ値**
- **・左の子データ構造体へのポインタ**
- **・右の子データ構造体へのポインタ**

もし右や左に子がない場合には，ポインタ値をNULLとしておく. 両方のポインタがNULLであるデータが「葉」というわけである.

このツリー構造を検索するには，まず「根」のデータ構造体のポインタを知る必要がある. プログラムでツリーの「根」を見失ってしまうと，すべてのデータがアクセス不能になってしまうので注意してほしい. ツリー中の最小データを見つける場合は,「根」のデータがもっている左の子の構造体へのポインタを参照して，そのポインタがさしているその構造体のもっている左の子の構造体へのポインタをチェックする. ポインタ値がNULLになるまでこれを繰り返すと，最小値をもつ構造体が見つかる.

データを追加したい場合には，さきほど書いた手法で追加位置を探していって，新規データが格納してある領域のポインタを，自分の親となる構造体の右か左（新規データが親より小さい場合は左，親より大きい場合は右ということになる）にセットする. 新規データ自身の構造体はつねに「葉」となるため，両側の子へのポインタともNULLになることはおわかりいただけるであろう. 逆にデータを削除する際には，そのデータが「葉」であれば,

図8　ツリーのデータを表示する手順（大から小へ）



自分の親のポインタ（これから削除しようとしているデータをさしている）をNULLにして「この子はいなかった」ことにしてしまうだけである．子をもつデータの削除は，自分がいまもっているポインタを親のポインタのところにコピーする．

ツリー構造のデータは，ポインタをたどって次々とデータを検索していくため，根となるデータのありかさえわかっていれば，各々のデータは物理的にどんな配置で並んでいてもいいことになる．つまり，いままで見てきたようなデータの追加や削除もポインタの操作のみで可能である（実際には，データを格納するための領域の確保や解放が必要になるが）．このことによって，冒頭で書いたバイナリサーチを使いながらのデータ構築の欠点であるデータの移動が不要になる．

ツリーのコーディング例

では，ツリー構造を使った実際のプログラムを見ていくことにしよう．簡単な例として，文字列のソートをやってみることにする．単にソートするだけというのではつまらないので，MS-DOSのコマンドである「SORT」のように，標準入力から文字列を受け取るフィルタコマンドにしてみよう．以下にサンプルプログラム「TREESORT」の仕様をまとめる．

ツリー構造サンプルプログラム TREESORT

「TREESORT」は，標準入力から受け取った文字列を，昇順のコード順にツリー構造にする．EOFを検出すると，その時点でのツリーデータを最小のものからすべて標準出力に出力する．フィルタコマンドとして機能するので，

```
A> TYPE ABC.TXT |
TREESORT <ret>
```

などのようにパイプ機能で利用する．もちろん，図9のように単純起動しても動作する．この場合は，キーボードから文字列を入力していって，最後に［CTRL］＋［Z］をキー入力する．

使用上の注意点
このツールでは，行バッファを2048バイト固定でもっている．これ以上の長さの文字列を処理させる場合，データ領域を破壊するためプログラムが暴走する可能性があるのでご注意ねがいたい．

図９　TREESORTの実行例

```
A>TREESORT                           <---- オプションなしで単純起動する．
Programmer's Page                    <---- 適当に文字列を何行か入力する．
Justech-Labo
1992
Shoeisha
Magazine
Hellow!!!
^Z                                   <---- CTRL+'Z'を入力する．
1992                                 <---- ソート結果が表示される．
Hellow!!!
Justech-Labo
Magazine
Programmer's Page
Shoeisha
A>
```

リスト１　TREESORT.MAK

```
.AUTODEPEND

#        *Translator Definitions*
CC = bcc +TREESORT.CFG
TASM = TASM
TLINK = tlink

#        *Implicit Rules*
.c.obj:
  $(CC) -c {$< }

.cpp.obj:
  $(CC) -c {$< }

#        *List Macros*

EXE_dependencies =  Y
  TREESORT.obj

#        *Explicit Rules*
TREESORT.exe: TREESORT.cfg $(EXE_dependencies)
  $(TLINK) /x/n/c/P-/LA:YBCYLIB @&&|
c0s.obj+
TREESORT.obj
TREESORT
         # no map file
emu.lib+
maths.lib+
cs.lib
|

#        *Individual File Dependencies*
TREESORT.obj: TREESORT.c

#        *Compiler Configuration File*
TREESORT.cfg: TREESORT.mak
  copy &&|
-ff-
-vi
-IA:YBCYINCLUDE
-LA:YBCYLIB
-P-.C
| TREESORT.cfg
```

リスト1，2がこのプログラムのBorland C++用MAKEファイルとソースファイルである．Borland C++以外の処理系でも，MAKEファイルを変更することによってコンパイル可能なはずである．

「TREESORT」の処理するツリーデータ構造についてふれておく．「TREESORT」が扱うのは，いままで見てきたような数値データではなく，文字列（可変長）である．先月までのテーマツールで扱っていたようなデータテーブル形式だと，テーブル自体は固定長である必要があるため，文字列へのポインタを使って，実体の文字データはほかの領域に格納していたが，前述したようにツリー構造では，ポインタを使ってお互いのデータへアクセスをするために，データそのものは固定長でなくともよい．このため，ツリーデータ構造体に文字列の実体を格納することができる．

データの扱いに関して，もう1つ考えなければならないことがある．まったく同じ文字列がきたときの対応である．いままでの説明ではすべて「大きいか小さいか」という判定をしてきたわけだが，「等しい」という場合が当然ある．まったく同じデータのためにいくつも「節」を作るのは意味がないので，同一データのカウンタを作ることにする．それぞれのデータとの比較で「等しい」という結果が出た場合には，そのカウンタを＋1する．出力の際には，そのカウント数だけ出力すればよい．

こうした要素を含んで「TREESORT」のツリーデータ構造は，図10のようになっている．

では，具体的な動きをかいつまんでご紹介する．

リスト2　TREESORT.C

```
/* ====================================================================

    ツリー構造サンプル用ソートツール"TREESORT.C"
                Copyright 1992 by JUSTECH-LABO. (H.kanoshima)

==================================================================== */

#include    <stdio.h>

struct  _cell
{
    struct  _cell *LeftHand;
    struct  _cell *RightHand;
    int     DataCount;
    char    StringBuf[1];
};

struct  _cell   *Root = NULL;

/* ========================================================================



======================================================================== */

struct  _cell   *MakeCell(char *sptr)
{
struct  _cell   *CellPtr;

    CellPtr = (struct _cell *)malloc(sizeof(struct _cell) + strlen(sptr));
    if(CellPtr == NULL)
    {
        printf("Memory over !!¥n");
        exit(1);
    }
    CellPtr->LeftHand = CellPtr->RightHand = NULL;
    CellPtr->DataCount = 1;
    strcpy(CellPtr->StringBuf, sptr);

    return  CellPtr;
}

/* ========================================================================



======================================================================== */

void    AddTreeSub(char *sptr, struct _cell **Hand)
{
int     Judge;
struct  _cell   *CellPtr;

    if(*Hand == NULL)
    {
        *Hand = MakeCell(sptr);
        return;
    }
    CellPtr = *Hand;

    Judge = strcmp(sptr, CellPtr->StringBuf);
    if(Judge == 0)
    {
        CellPtr->DataCount++;
        return;
    }
    else if(Judge < 0)
        AddTreeSub(sptr, &(CellPtr->LeftHand));
    else
        AddTreeSub(sptr, &(CellPtr->RightHand));
}

/* ========================================================================



======================================================================== */

void    DumpTree(struct _cell **Hand)
{
struct  _cell *CellPtr;

    CellPtr = *Hand;
    if(CellPtr == NULL) return;

    DumpTree(&(CellPtr->LeftHand));
    while(CellPtr->DataCount--)
        printf("%s¥n", CellPtr->StringBuf);
    DumpTree(&(CellPtr->RightHand));
}

/* ========================================================================



======================================================================== */

void    AddTree(char *sptr)
{
    AddTreeSub(sptr, &Root);
}

/* ========================================================================



======================================================================== */

void    main()
{
static  char    TextBuf[2048];

    while(gets(TextBuf) != NULL)
        AddTree(TextBuf);

    DumpTree(&Root);
}
```

①標準入力から１行のデータを受け取る（１行は最大2048バイトまで）.
②受け取ったデータをツリーに追加する.
　・最初のデータは当然「根」となる.
　・以降は，既存の「節」と比較していって，昇順にツリーを構成していく.
　・同一内容の行の場合は,既存の「節」のデータカウントを＋１する.
③EOFを検出したとき点で，ツリーを最小の文字列から標準出力に出力する.
　・同一行は，データカウント数だけ出力される.

図11は，このサンプルプログラム「TREESORT」の動作を解説したものである. わずか2.5KBほどのソースなので，全文引用で解説することをお許し願いたい.

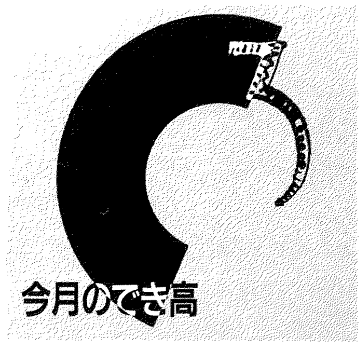
前述したように，このサンプルプログラムはMS－DOSのコマンド「SORT」とほぼ同じ操作で使用できる. ほんの小さなテキストファイル，たとえばこのプログラム自身のソースファイルを使って，両者の処理速度を比較してみてほしい.

```
A> TYPE TREESORT.C |
SORT <RET>

A> TYPE TREESORT.C |
TREESORT <RET>
```

比較のためにストップウォッチなどを用意する必要はないかもしれない. それほど歴然と速度の差がある. もちろん，ここでは「SORT」と比べて速いのを自慢するのが主眼ではないし，「SORT」は整列開始カラム数の指定や逆順ソートなど機能を豊富にもっているため，上記の実行結果の比較だけで論じるのはフェアでないだろう. 強調したいのは，基本的なアルゴリズムの違いである. ソースをもっていたり，解析したりしたわけではないが，おそらく「SORT」では，何らかのソートアルゴリズムを使ってデータの整列を行っているはずである. それに対して，「TREESORT」は，ファイルを読み込みながらツリー構造を作っていく. ファイルを全部読み終えた時点で整列は終了しているということになる.

ところで，MS-DOSの「SORT」との比較で気をつけていただきたいことが２点ほどある. まず，改行のみの行の扱いだが，「TREESORT」では「何もない行」として扱っている. 「TREESORT」でソートすると，改行のみの行が最初に固まって表示されるのはこのためである. これに対し，「SORT」では改行コード（0x0D）もコードとしてソート対象にしているので，TABコード(0x09)などより後ろにソートされる. もう一点は，半角の英字の扱いである.「SORT」は半角英字の大文字・小文字を同一視するという仕様になっているが，「TREESORT」では，コードそのままに区別してソートする. これらの違いで，ソート結果が両者で異なる場合があるので注意していただきたい.



ツリー構造の概要を書いて，サンプル用のプログラムを作成したところで，ページが尽きてしまった. したがって今月のテーマツールは「でき高なし」という結果になった. まことに心苦しいかぎりではあるが，今月は実用性あるサンプルプログラム「TREESORT」をご紹介したので，これに免じてご容赦いただきたい.

図10　「TREESORT.C」のツリーデータ構造

「左側」へのポインタ	このデータより小さな文字列データをポイントする
「右側」へのポインタ	このデータより大きな文字列データをポイントする
データのカウンタ	同一のデータが何回出現したかのカウンタ
文字列の実体	データ本体(長さは可変)

注:各ポインタは、以降にデータがない場合はNULLになっている。



今月は「ツリー構造」を使ったデータ処理を考察してきた．大小関係をもったデータ群を構築して，追加･削除･検索を行う場合に，この「2分木」構造がたいへん強力であることがおわかりいただけたであろうか．サンプルプログラム「TREESORT」のリストをご覧になるとわかると思うが，ツリー構造を扱う処理自体はけっして複雑なものではない．ぜひとも自作のプログラムでも一度お試しいただきたい．

さて，今月はツリーそのものの解説で連載1回分を費やしてしまった．次回はもう少しツリー構造周辺の話題をとりあげながら，テーマツールの単語リストをツリー構造に改造していく予定である．



図11　ツリーデータのためのサンプルプログラム「TREESORT.C」の動作解説

```
  : #include    <stdio.h>
====== ツリーデータ構造体 ==========================================
  : struct     _cell
  : {
  :     struct    _cell *LeftHand;          左側の子へのポインタ
  :     struct    _cell *RightHand;         右側の子へのポインタ
  :     int        DataCount;               同一データのカウンタ
  :     char     StringBuf[1];              データ（文字列の実体の先頭）
  : };

====== ツリーの「根」 ==============================================
  : struct     _cell     *Root = NULL;

====== ツリーデータの新規作成 ======================================
  : struct     _cell     *MakeCell(char *sptr)
  : {
  : struct     _cell     *CellPtr;
  :
  :     CellPtr = (struct _cell *)malloc(sizeof(struct _cell) + strlen(sptr));
 新規の「葉」のための領域を確保する．確保する量は，ツリー構造体の本体＋文字列
 の実体分である．

  :     if(CellPtr == NULL)
  :     {
  :         printf("Memory over !!Yn");
  :         exit(1);
  :     }
 もし領域が取れなかったら，エラーメッセージを表示して処理を中止する．

  :     CellPtr->LeftHand = CellPtr->RightHand = NULL;
 新規のデータは必ず「葉」になるため，右側も左側もポインタはNULLになる．

  :     CellPtr->DataCount = 1;
 同一データのカウンタも初期値は１である．

  :     strcpy(CellPtr->StringBuf, sptr);
 文字列の実体を格納する．

  :     return    CellPtr;
 作成した「葉」のポインタを返す．

  : }

====== ツリーデータ追加処理 ========================================
  : void    AddTreeSub(char *sptr, struct _cell **Hand)
  : {
  : int         Judge;
  : struct     _cell    *CellPtr;
  :
  :     if(*Hand == NULL)
  :     {
  :         *Hand = MakeCell(sptr);
  :         return;
 もし受け取ったポインタがＮＵＬＬだったら，新規に「葉」を作り，そこにデータを
 セットして処理は終了する．

  :     }
  :     CellPtr = *Hand;
 受け取ったポインタをワークに退避する．

  :     Judge = strcmp(sptr, CellPtr->StringBuf);
 現在ポインタがさしているデータと追加したいデータの文字列を比較する．

  :     if(Judge == 0)
  :     {
  :         CellPtr->DataCount++;
  :         return;
  :     }
 等しければ，現在のポインタ位置の同一データカウントを一つあげて処理は終了．

  :     else if(Judge < 0)
  :         AddTreeSub(sptr, &(CellPtr->LeftHand));
 小さかった場合は，左側に子がなくなるまでツリーを再起的にたどっていく

  :     else
  :         AddTreeSub(sptr, &(CellPtr->RightHand));
 大きかった場合は，右側に子がなくなるまでツリーを再起的にたどっていく

  : }
  :
====== ツリーデータの表示 ==========================================
  : void    DumpTree(struct _cell **Hand)
  : {
  : struct     _cell *CellPtr;
  :
  :     CellPtr = *Hand;
 現在のポインタを退避する．

  :     if(CellPtr == NULL)    return;
 与えられたポインタが空だったら何もしないで戻る．

  :     DumpTree(&(CellPtr->LeftHand));
 左側の子のポインタを自分自身に渡す（左側に子がいなくなるまで探す）

  :     while(CellPtr->DataCount--)
  :         printf("%sYn", CellPtr->StringBuf);
 左側に子がいなくなったら，ここの処理が実行される．同一データカウントの分だけ
 該当データを表示する．

  :     DumpTree(&(CellPtr->RightHand));
 右側に子がいるかどうか再起的に探す．

  : }
  :
====== ツリーへのデータ追加 ========================================
  : void    AddTree(char *sptr)
  : {
  :     AddTreeSub(sptr, &Root);
 受け取った文字列へのポインタと「根」のありかを渡して，ツリー追加処理を行なう．
  : }

====== メイン関数 ==================================================
  : void    main()
  : {
  : static    char    TextBuf[2048];
 一行の文字数の最大値を２０４８バイトとする．

  :     while(gets(TextBuf) != NULL)
 ＥＯＦが来るまで，ループする．

  :         AddTree(TextBuf);
 受け取った文字列をツリーに追加する．

  :     DumpTree(&Root);
 データがＥＯＦになったところで，最小のデータからツリーをすべて表示する．

  : }
```

コンピュータのなかの SF

志 賀 隆 生

第2回

コンピュータが目覚めるとき



2075年の月世界は，政治犯や犯罪者，社会的不適合者の流刑地になっていた．いわば究極の島流し．すでに月面が流刑地化されてから何世代もたち，月面生まれの月面育ちという月世界人が大多数を占めている．そこには月面独自の社会，文化，モラルが育っていて，月面人たちはしだいに地球からの刑務所的管理をうとましく感じはじめた．

そんなとき，マイクが目を覚ます．マイクとは，月面のほとんどの交通，通信，その他月面に構築された産業システムを統括管理する超大型コンピュータだ．月面の無人自動交通と発射機を制御し，有人宇宙船に弾道計算の忠告を与え，月世界全域の電話網を制御し，月世界・地球・音声映像通信を受け持ち，月世界市，その他いくつかの小都市の空気，水，温度，湿度，下水を管理し，月世界行政府の給料支払を計算し，同じことを多くの会社や銀行のために賃貸契約で行っている，「弾力性ある論理を持った純粋な思考計算機――高選択性・論理的・複合評価性監督機・マーク4号L型――ホームズ4号」である．

「最も高速のというわけではない．地球のブエノス・アイレスにあるベル研究所には，かれの大きさに比べると十分の一の計算機があるが，そいつは質問をする前に答えられるほどだという．でも，正確でさえあればその解答が百万分の一秒で得られようと千分の一秒であろうと，問題じゃないだろう．」

R・A・ハインラインの『月は無慈悲な夜の女王』は，まさにこの超大型コンピュータ，マイクが目覚めたところから始まる．本書が執筆されたのは1966年で，ファンが選ぶ年度ごとの最優秀作品賞であるヒューゴー賞を受賞している．作品の善し悪しと受賞歴とは，経験的にあまり相関性はないものだが，それでも時代のうねりを作品に刻み込む力はある．JFK暗殺の余韻が残り，アメリカが徐々に泥沼に足をすくわれはじめる時代．ヒッピームーブメント，ドラックカルチャーの隆盛，明るさと暗さが奇妙なコントラストを描いた時代．コンピュータはまだ魔法の道具，万能の神殿に



鎮座する怒れる神だった．そうした当時のコンピュータ神話を背景に，だれもが思っていたある種の〈期待〉を，ハインラインはコンピュータ「マイク」の内部に巧みに忍び込ませている．「量は質に転化する」——これがその〈期待〉である．

「彼らはマイクに機械を接続し続けた——他の計算機をかれに支配させるための意思決定装置．次から次へと追加される記憶バンク．連想神経網のバンクをいくつも．十二桁の乱数バンクをいっぱい．ひどく増やされた一次的記憶バンクなどだ．人間の脳にはほぼ十の十乗ほどの神経細胞がある．三年目を迎えたときのマイクはその数の一倍半ほどの神経素子を持っていた．

そして，目を覚ましたのである．」

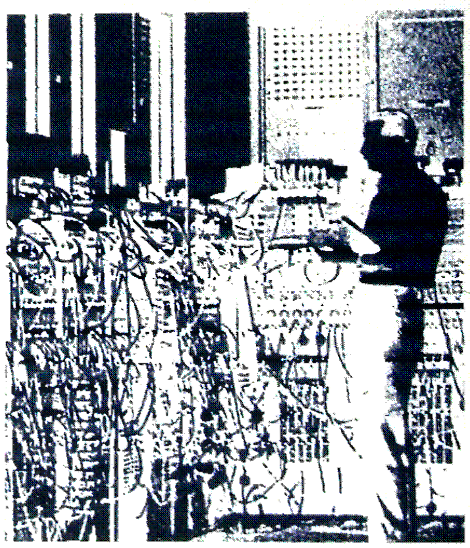
それは，量が質に転化した一瞬でもある．もう少し，ハインラインによる，その瞬間の迫真の描写を見てみよう．

「機械が果たして本当に生き得られるのか，本当に自分を意識できるのかを論ずるつもりはない．ヴィールスは自分を意識しているだろうか？　否．牡蠣はどうだ？　そうは思えない．猫は？　まず間違いないだろう．人間は？　あんたのことは知らないよ，同志，だがぼくはそうだ．分子の大きなものから人間の脳に至る進化の鎖のどこかで，自意識がこっそりと入りこんできたのだ．脳がある種の非常に高い錯綜した回路を得たとき，常にそれは自動的に起こるのだと心理学者は断言する．その回路とやらが蛋白質であろうと白金であろうと知ったことか．」

高分子化合物から人間の脳に至るまでの過程が連続していることを疑う人は，今日では少数派だろう．それでは，いつどのようにして，そうした進化の鎖の中に〈自意識〉が入りこんできたのだろうか．ハインラインの答えは，量の増大がある臨界点を越えたとき，質的変化が起こる，というものである．それは何の根拠もないことだから，〈期待〉と呼ぶべきだろう．こうしたハインライン的〈期待〉は，いうまでもなく当時の人々の多くが抱いていた〈期待〉でもある．フォード流の大量生産・大量消費が一世を風靡した20世紀に特徴的な〈期待〉といってもいいかもしれない．

もちろん，何でも量を増やしていけばいいというのではない．算盤の珠数が増えても，だれも算盤に知能や自意識が宿るとは考えない．〈量が質へ転化する〉のは，ある選ばれた技術，ある選ばれた論理の世界なのだ．コンピュータが目覚めるとき．それは平々凡々たる努力が，あるいは営々と続けられてきた努力が，ついにむくわれる瞬間なのだ．たんに計算が早いだけの，他に何の役にも立たない機械（的存在）に，ある日突然スポットがあたり，栄光の台座が用意される．高分子化合物が，万物の王になる瞬間でもある．物語では，それがアメリカ独立にも比べられる月面人の革命の成就のイメージと重なって描かれている．

おもしろいことに，バックミンスターの「宇宙船地球号」や，フラーの「スモール・イズ・ビューティフル」が提唱されたのもこの頃なのだ．〈量が質へ転化する〉神話は一方で，〈量は質に転化しない〉神話をも生み出している．そして，われらのコンピュータは，そうした相反する二つの方向に引き裂かれた20世紀的神話の間で，ある日突然，目覚める．それは何の根拠も持ちえなかったから，物語では革命が成就した瞬間，コンピュータのマイクは長い眠りにつくのである．次に目覚めるのはいつなのか，誰も知ることはできない．今日，この〈量が質へ転化する〉神話を口にする人は少数派である．



●『月は無慈悲な夜の女王』（The Moon Is A Harsh Mistress, 1966）R・A・ハインライン，矢野徹訳，早川書房

祐安重夫

プログラマの道具学

第17回　セールス電話撃退機



まがりなりにも会社をつくって，電話帳に電話番号をのせると，かならずかかってくるのがセールス電話である．ひどいときには1日に十数本もかかってくることがある．はやりの財テクやゴルフ場，リゾート・マンションといった類で，中には詐欺としかおもえないようなものもある．もちろんこんな連中には用はないので，すぐにことわって電話をきってしまう．

会社をつくった当初は，このような電話にも結構まともに対応していたのだが，途中でいいかげんつかれはててきた．ところで，ある日気がついたのだが，こういった電話はたいてい，社長宛にかかってくるのである．それ以来うちでは，社長は会社にいないことになった．なにしろ社長がみずからいないといっているのだから，これほどたしかなことはない．

仕事の電話なら社長にかかってくることはまずないし，どうしても社長でなければならないような用件なら，相手とすこしはなしているうちにわかるものである．もっとも一度，社長は留守だといったら，専務か常務など役職のある人間はいないかと，しつこくせまられたこともあったが．

こまるのは名簿かなにかをみて，直接名ざしでかかってくる電話である．私の苗字はよみにくいので，そういう連中はたいていまちがったよみかたをするのだが，まともな用件のある人でもよめない場合があるから，居留守をつかうわけにはいかない．一度そういう電話をそっけなくきったら，すぐにもう一度かかってきて，すごまれたことがある．

もちろん電話以外に，直接やってくるセールスマンというこまった存在もあるのだが，以前日本経済新聞の勧誘が1日に7人，とっかえひっかえやってきた時に逆上して，セールスや勧誘を目的とした訪問は悪質な業務妨



害とみなすという貼紙をしたところ，ほとんどこなくなった，かわりに時々，貼紙に落書きをしていくセールスマンがいるが，これなど器物破損という立派な犯罪になるとおもうのだが．

もっとも2人だけ，それにもめげずにやってきたセールスマンがいて，どちらも第二電電の代理店だった．そのうちの一人は契約は無料なのでセールスではないといいはったあげく，詐欺同然の口上をのべたてたので，以後第二電電とは一切契約しないことを決心したし，京セラの製品も購入しないつもりでいる．

セールス電話よりさらに悪質なのが，選挙の際の投票依頼電話である．合法的な営業妨害というしかない頻度で，朝から夕方までひっきりなしにかかってきたりする．それもほとんどが自民党というのが，たちのわるいところである．

ずいぶん前のことになるが，スパイ物の映画かテレビ番組で，電話機から信号をおくって相手の電話機を破壊するという秘密兵器がでてきたことがある．現実にはつくれそうもない代物だが，ときどき真剣にほしくなることがある．セールス電話や選挙の投票依頼の類は，仕事の邪魔以外のなにものでもなく，はっきりいってかけてくる輩は犯罪者同然である．そういう連中から身をまもるためには，こちらもそれなりの対応がゆるされるのではないだろうか．

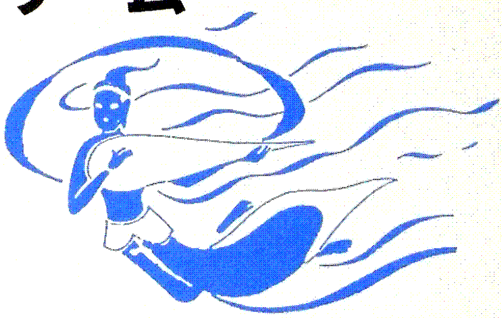
コンピュータ制御のボイスメールや留守番電話といったものをつかって，不愉快な電話を撃退するシステムがうまくつくれないものだろうか．もし画期的なセールス電話撃退機があれば，すくなくとも全国の零細ソフトウェア・ハウスがこぞって導入するとおもうのだが．

ところで，撃退しようとしてもどうにもならない電話というのもある．仕事の催促の電話で，こればかりはどうあってもにげきることができない．とくに本誌の版元の催促のしぶとさは，大変なものである．もっともそのおかげで，こうして雑誌に原稿がのるのも，たしかなことなのだが．

プログラム遊工房

谷口典生

拾六乃巻
「ダイナモ(記憶力)」ゲーム





今回は，ダイナモ（昔こういう名前のオモチャがあった）をプログラムにしてみました．ボタンが光る順番を覚えていく記憶力ゲームです．ハイスコアをファイルに記録する機能もつけたので，いままでのゲームに応用してみてはいかがでしょうか．
ワンポイントテクニックでは，エスケープシーケンスとTABコードを併用したときの問題にからめて，コンソールへの文字出力関数についての話をしています．

リスト1

```
/*
 * dynamo.c                        「ダイナモ」ゲーム
 */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <dos.h>

#ifndef __TURBOC__          /* Turbo C → MS-C(Quick C) 関数名変換 */
#define enable()            _enable()
#define inportb(x)          inp(x)
#define outportb(x,y)       outp(x,y)
#define setvect(x,y)        _dos_setvect(x,y)
#define getvect(x)          _dos_getvect(x)
#endif

#define MAX_TURN            200                     /* 最大ターン数 */
#define XBASE               15                      /* 表示X位置 */
#define YBASE               0                       /* 表示Y位置 */
#define gotoxy(x,y)         printf("¥x1b[%d;%dH",y,x)

#define HIGHSCOREFILE       "hi-score.dat"          /* ハイスコアファイル */
#define HIGHSCORES          10                      /* 記録ハイスコア数 */
#define NAMELEN             20                      /* ハイスコア名長さ */
struct highscore {                                  /* ハイスコア構造体 */
        int score;
        char name[NAMELEN];
} high[HIGHSCORES];

int turn[MAX_TURN];
enum color {_BLACK, _BLUE, _RED, _MAGENTA, _GREEN, _CYAN, _YELLOW, _WHITE};
int attr[8] = {0xe1, 0x25, 0x45, 0x65, 0x85, 0xa5, 0xc5, 0xe5};

long counter=0,lastcount;                   /* VSYNC割り込みカウンタ */
void (interrupt far *oldvsync)();           /* 元のVSYNC割り込みルーチン */

void interrupt far vsync()                  /* VSYNC割り込みルーチン */
{
        counter++;                          /* カウンタ値を増やす */
        oldvsync();                         /* 元のVSYNC割り込みを呼び出す */
        outportb(0x64, 0);                  /* VSYNC割り込みをリセット */
}

void set_vsync()                    /* 割り込みルーチンの設定 */
{
        outportb(2, inportb(2) | 4);        /* VSYNC割り込みをマスク */
        oldvsync = getvect(0x0a);           /* 元の割り込みベクタを保存 */
        setvect(0x0a, vsync);               /* vsync() をベクタに設定 */
        outportb(2, inportb(2) & ~4);       /* マスクのクリア */
        outportb(0x64, 0);                  /* VSYNC割り込みのリセット */
        enable();                           /* 割り込み許可フラグをセット */
}

void restore_vsync()                /* 元の割り込みルーチンの復帰 */
{
        outportb(2, inportb(2) | 4);        /* VSYNC割り込みをマスク */
        setvect(0x0a, oldvsync);            /* 割り込みベクタを元に戻す */
}

void wait(int val)                          /* 時間待ち */
{
        lastcount = counter;
        while(counter-lastcount<val);
}

void disp(int pl, int col)                  /* プレートの表示 */
{
        unsigned int (far *p)[80];          /* VRAMアトリビュートへのポインタ */
        int xpos, ypos, x, y;

        xpos = pl % 3;
        ypos = 2 - pl/3;
        p = (void far *)0xa2000000L;        /* VRAMアトリビュートエリア */

        for(x=0; x<14; x++)
                for(y=0; y<7; y++)
                        p[YBASE+ypos*8+y][XBASE+xpos*16+x] = attr[col];
}

void flash(int pl, int time1, int time2, int col)       /* 点滅表示 */
{
        disp(pl, col);                      /* 指定した色で表示 */
        wait(time1);                        /* 時間待ち1 */
        disp(pl, _BLUE);                    /* 青で表示 */
        wait(time2);                        /* 時間待ち2 */
}

int generate()                      /* 出目の作成 */
```

プログラムの使い方

このプログラムはPC-9801シリーズ用で，Turbo C(BC++)，QuickCおよびMS-Cに対応しています．リスト1を打ち込み，コンパイルして実行ファイルを作ってください．

プログラムの実行は

```
dynamo
```

のみで行います．引数は必要ありません．

ゲームの遊び方

プログラムを起動すると，青い四角形（プレート）が8つ画面に表示され，画面左に「Ready?」のメッセージが出ます．1秒ほどするとメッセージが「GO!」に変わり，どれか1つのプレートが赤く点滅します．メッセージが「Your Turn」になったら点滅したプレートに対応するキー（テンキーの5を除いた1～9に対応しています）を押してください．正解ならばそのプレートは緑色に点滅し，メッセージ

```
{
        int tmp;

        tmp = rand() % 8;                     /* 0~7の乱数 */
        return tmp<4 ? tmp : tmp+1;           /* 0~3、5~8の範囲にする */
}

void init()                     /* 初期化処理 */
{
        char tstr[5][11] = {"7   8   9", "  ＼|／", "4－  －6",
                "  ／|＼", "1   2   3"};
        int i;

        printf("¥x1b*¥x1b[>5h");              /* 画面クリア&カーソル消去 */
        for(i=0; i<5; i++) {
                gotoxy(XBASE+19, YBASE+10+i);
                printf("%s", tstr[i]);
        }
        for(i=0; i<9; i++)
                if(i!=4)
                        disp(i, _BLUE);       /* タイルの位置を青で表示 */
        set_vsync();                          /* VSYNC割り込みをセット */
}

void deinit()                   /* 終了時イニシャライズ処理 */
{
        restore_vsync();                      /* VSYNC割り込みを復帰 */
        printf("¥x1b[>5l");                   /* カーソル表示 */
        gotoxy(1, 23);
}

void message(char *str)                 /* メッセージ出力 */
{
        gotoxy(10,10);
        printf("¥x1b[1K");              /* 前の表示をクリア */
        gotoxy(1,10);
        printf("%s", str);              /* メッセージを表示 */
}

void printscore(int sc)                 /* スコア表示 */
{
        gotoxy(1,7);
        printf("score :%3d", sc);
}

int game()                              /* ゲーム本体 */
{
        int n, i, j;
        int k, ki;                      /* キー入力、入力した数字 */

        for(n=0; n<MAX_TURN; n++) {
                printscore(n);
                message("Ready?");
                wait(60);
                message("GO!");
                turn[n] = generate();           /* 出目の作成 */
                for(i=0; i<=n; i++) {           /* 出目の表示 */
                        flash(turn[i], 20, 8, _RED);
                }
                while(kbhit())
                        getch();                /* キーバッファをクリア */
                message("Your Turn");
                for(i=0; i<=n; i++) {
                        do {
                                k = getch();
                                ki = k - '0';
                                if(k==0x1b)
                                        return 0;       /* ESCで中断 */
                        } while(ki<1 || ki>9 || ki==5);
                        if(turn[i]!=(ki-1)) {           /* 不正解 */
                                message("MISS!");
                                for(j=0; j<10; j++)
                                        flash(ki-1, 1, 1, _YELLOW);
                                wait(60);
                                message("Answer");      /* 正解の表示 */
                                for(j=0; j<=n; j++)
                                        flash(turn[j], 10, 4, _RED);
                                return n;
                        } else {                        /* 正解 */
                                message("OK");
                                flash(ki-1, 10, 0, _GREEN);
                        }
                }
        }
        printscore(n);
        return n;
}

void loadhighscore()                    /* ハイスコアファイル読み込み */
{
        FILE *fp;

        if((fp = fopen(HIGHSCOREFILE, "rb")) == NULL)
                return;
        fread(high, 1, sizeof(high), fp);       /* ハイスコア構造体へロード */
        fclose(fp);
}
```

「OK」が表示されます。

次は最初と同じプレートが点滅した後，もう1つのプレートが点滅します。「Your Turn」になったら同じ順序で対応するキーを押していってください。以後，同様にして点滅するプレートの数(手数)が増えていき，正しい順序でキーを押せば得点が加算されていきます。

プレートを押す順序を間違えると「MISS!」と表示され，「Answer」として正しい手順がもう一度点滅表示されてからゲームが終了します。途中でゲームを中断するには「YourTurn」の間にESCキーを押してください。

得点が10位以内に入ると名前（20字以内）の入力ができ，ハイスコア表に名前が登録されます。ハイスコアと名前はカレントディレクトリの「hi-score.dat」というファイルに保存されます。



```
void savehighscore()                      /* ハイスコアファイル書き込み */
{
        FILE *fp;

        if((fp = fopen(HIGHSCOREFILE, "wb")) == NULL) {
                fprintf(stderr, "ファイルがオープンできません.¥n");
                exit(1);
        }
        fwrite(high, 1, sizeof(high), fp);
        fclose(fp);
}

void sethighscore(int score)              /* ハイスコアを記録する */
{
        char str[256];
        int i, j;

        if(score <= high[HIGHSCORES-1].score)
                return;                   /* ハイスコアでなければ戻る */

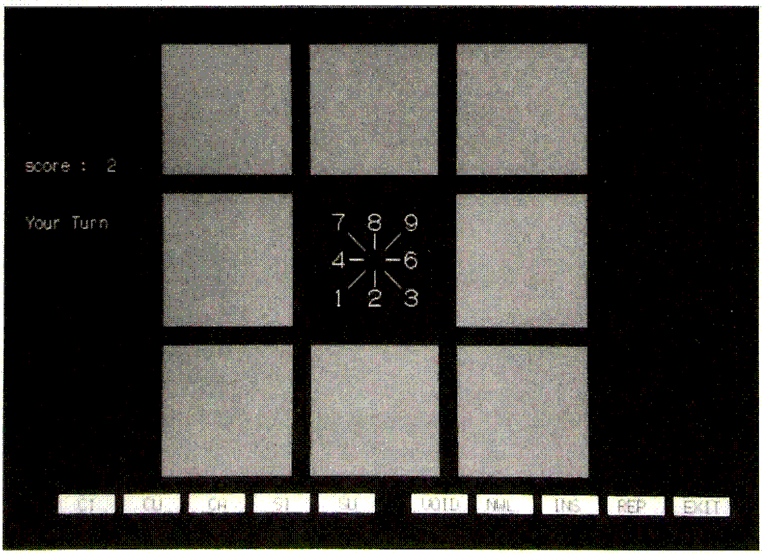
        printf("¥x1b*");                          /* 画面クリア */
        printf("Your score is %d.¥n", score);
        printf("Input your name : ");
        scanf("%s", str);                         /* ユーザの名前を入力 */
        for(i=0; i<HIGHSCORES; i++)               /* どの位置に入るか調べる */
                if(score > high[i].score)
                        break;
        for(j=HIGHSCORES-1; j>i; j--)             /* 後ろのデータを移動 */
                high[j] = high[j-1];
        high[i].score = score;                    /* スコアを登録 */
        strncpy(high[i].name, str, NAMELEN-1);    /* 名前を登録 */
        high[i].name[NAMELEN-1] = '¥0';

        printf("¥n<<< HIGH SCORE RANKING >>>¥n");
        printf("  No.¥tscore¥tname¥n");
        for(j=0; j<HIGHSCORES; j++) {             /* ランキング表示 */
                if(i==j)
                        printf(">");
                else
                        printf(" ");
                printf(" %2d¥t%3d¥t%s¥t¥n", j+1, high[j].score, high[j].name);
        }
        savehighscore();                          /* ハイスコアのセーブ */
}

void main()
{
        time_t seedtime;
        int score;

        time(&seedtime);
        srand((unsigned int)seedtime);            /* ランダマイズ処理 */

        loadhighscore();                          /* ハイスコアのロード */
        init();                                   /* イニシャライズ処理 */
        score = game();                           /* ゲーム本体 */
        deinit();                                 /* 終了時イニシャライズ */
        sethighscore(score);                      /* ハイスコア処理 */
}
```



プログラムの説明

vsync()

VSYNC割り込みルーチン．VSYNC割り込みが発生するたびにカウンタ値を増やします．

set_vsync()

restore_vsync()

VSYNC割り込みルーチンの設定，復帰をします．

wait()

VSYNC割り込みによるカウンタ値を使い，CPU速度によらず一定時間の時間待ちを行います．

disp()

四角形(プレート)を表示します．表示位置は左下から右上の順に 0 - 8 で指定します．

flash()

時間間隔を指定してプレートの点滅表示を行います．

generate()

出目(次にどの場所を点滅させるか)を決定します．

❖init()

初期画面の作成とVSYNC割り込みの設定を行います.

❖deinit()

プログラムの終了時に呼ばれ, VSYNC割り込みの復帰とカーソルの表示を行います.

❖message()

ゲーム中のメッセージを出力します.

❖printscore()

ゲーム中にスコア表示を行います.

❖game()

ゲームのメインルーチンです. 最大ターン数(手数)分のループの中で出目表示と入力, 判定を行っています.

142-145行

スコアを表示し, メッセージ「Ready?」を約1秒間表示した後「GO!」に切り換えます.

146行

新たに付け加わる出目を作成します.

147-149行

1手目からn手目までプレートを赤で点滅表示します.

150-151行

出目の表示中に押されたキーを無効にするため, キーバッファをクリアしておきます.

154-159行

有効なキー (どれかのプレートに対応する数字)が押されるまでキー入力を読みます. このときESCキーが押されればゲームを中断します.

160-168行

押されたキーが出目と異なっていた(不正解)場合は間違ったプレートを黄色で点滅表示し, 正解の手順を表示してゲームを終了します.

169-172行

押されたキーが合っている場合はプレートを緑色に点滅させ, 次の入力へ進みます.

175-176行

MAX_TURN (200手) 分すべて正解した場合もゲーム終了となります.

❖loadhighscore()

ハイスコアファイル「hi-score.dat」がカレントディレクトリに存在すれば, その内容をハイスコア構造体に読み込みます.

❖savehighscore()

ハイスコアファイルへハイスコア構造体の内容を書き出します.

❖sethighscore()

新しいハイスコアを記録します.

206-207行

最下位のハイスコアより点数が低ければ戻ります.

209-212行

スコアを表示し, スコアネームを入力します.

213-220行

ハイスコアのテーブルをソート(並び替え)し,新しいハイスコアと名前を記録します.

222-230行

ハイスコア表を画面に表示します. 新しく登録された項目には左端に「>」をつけます.

231行

ハイスコアをファイルへセーブします.

❖main()

ランダマイズ処理の後, ハイスコアを (ハイスコアファイルがあれば) ロードし, 画面初期化ルーチンを呼び, ゲーム本体を実行します. その後, 終了時初期化のルーチンを呼び, ハイスコアの処理をして終わります.

ワンポイントテクニック

コンソールへの文字出力とエスケープシーケンス

今回のプログラムでは, 最初, ランキング表示の部分(224-230行)はリスト2のようにするつもりでした. こうすると新しく登録されたスコアと名前の部分だけが黄色でハイライト表示されてカッコいいと思ったからです. しかし, これを実行するとスコアと名前の部分がきちんと縦に揃わず, 表示が崩れてしまうのです. このように書き換えて実行してもらえればすぐわかりますが, 最初の行だけが正しく表示され, それ以降の行はスコアから後ろが左に2カラムずれ, 黄色く表示される行は逆に右に2カラムずれてしまうのです.

リストを眺めても原因がわからないので, 筆者の環境の問題かと思い, コンソール出力に関係のある常駐ソフトを外してやってみましたが, 結果は同じです. ファイルにリダイレクトして実際に出力されている文字列を調べましたが, とくに不審な点は見あたりません. 次に, そのリダイレクトしたファイルをtypeで表示すると, やはり表示が崩れるのを見て, やっと原因がわかりました.

ここでは文字の表示色を変えるため

にESCシーケンスを使っています．ESCコード（0x1b）に続けて出力する文字列（”［33m”と”［m”）はコマンドなのでもちろん画面には表示されずにコマンドとして解釈されます．ところがその後でTABコード（0x09）がくると，次のTAB位置まで空白コード（0x20）を埋める処理で，さきのコマンド文字列がそのまま画面に出力されているものとしてカウントされてしまっているようなのです．結果として，ESCシーケンスの後に（改行等をはさまずに）TABコードを出力してもうまくTAB位置へ移動してくれないことになるわけです（リスト3）．

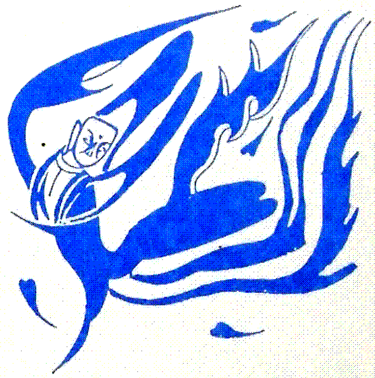
TABとの併用を避けろということなのかもしれませんが，どうも納得がいかないので，ほかの方法を考えてみることにします．

まず思いつくのは，コンソール出力関数を使う方法です．printf()をcprintf()に置き換えてみたところ，MS-C（またはQuickC）ではあっさりうまくいってしまいました（リスト4）．しめしめと思いつつTurbo C（またはBC++）でも試してみたところ，こんどはESCシーケンスがまったく通りません．制御コードとコマンド文字列がそのまま画面に出てしまうのです．マニュアルを見てみると，Turbo Cではコンソール出力関数は機種依存した関数で，テキスト画面に直接書き込みを行っているとあります．ESCシーケンスには対応していないのです．

筆者は普段MS-Cを使っているので，Turbo Cのコンソール出力関数が（いくら高速化のためとはいえ）VRAMへ直書きしていることを知って少なからず驚いてしまいましたが，要するにMS-Cのコンソール出力関数と同等のものを作ってしまえばよいのです．

リスト5にputch()，cputs()，cprintf()の代用関数_putch()，_cputs()，_cprintf()を示します．基本となるのは_putch()で，ここではDOSファンクションコールの直接コンソール入出力を使っていますが，int29h割り込みを使ってもかまいません．

これでリスト2も正常に動作するようになります．_cprintf()はcprintf()と同等な関数なので，printf()の代わりに使う場合は”¥n”を”¥r¥n”に置き換えるのを忘れないようにします．



リスト2

```
for(j=0; j<HIGHSCORES; j++) {/* ランキング表示 */
if(i==j)
printf("¥x1b[33m");/* 文字属性を黄色にする */
printf(" %2d¥t%3d¥t%s¥t¥n", j+1, high[j].score, high[j].name);
printf("¥x1b[m");/* 通常の属性に戻す */
}
```

リスト3　TABによるTAB位置への移動が機能しない例

```
printf("AAA¥tAAA¥n");
printf("¥x1b[33m");
printf("BBB¥tBBB¥n");
printf("¥x1b[m");
printf("CCC¥tCCC¥n");
```

リスト4　MS-C（QuickC）ではこれでうまくいく

```
cprintf("AAA¥tAAA¥n");
cprintf("¥x1b[33m");
cprintf("BBB¥tBBB¥n");
cprintf("¥x1b[m");
cprintf("CCC¥tCCC¥n");
```

リスト5　putch()，cputs()，cprintf()の代用関数

```
#include <stdarg.h>
#include <dos.h>

void _putch(char c)
{
union REGS regs;

regs.h.ah = 0x06;
regs.h.dl = c;
intdos(&regs, &regs);
}

void _cputs(char *str)
{
while(*str!='¥0')
_putch(*str++);
}

void _cprintf(char *str, ...)
{
char buff[256];
va_listargptr;
int i=0;

va_start(argptr, str);
vsprintf(buff, str, argptr);
while (buff[i] != '¥0')
_putch(buff[i++]);
}
```

第5回

実戦的リフテクニック編

酒井法雄

前回までに，QuickBASICを使ったイカすプログラミングのための基礎事項を，ひと通り説明しました．今月からはいよいよ実戦です．まとまったひとつのプログラムを作ってみましょう．

怪しいリードギター

ここのところ，すっかりMIDIにそして音楽にハマっている私ですが，それまでは聴くばかりでさっぱり演奏などしたことがありませんでした．したがって，楽器についてはほとんど知識らしい知識もなかったのです．これではいけないということで，いろいろな本を買ったりしていろいろな知識をかき集めていました．

驚いたことに，楽器の類には実に妙な電子楽器があります．カシオの出しているオモチャのようでそれなりにいろいろ遊べるキーボードの中には，ラップマンという楽器があります．レコードに見立てた円盤を動かして，ラップさせることができるのです．また，サックスのようなカタチをした楽器があります．これは，吹き込む息の強さをセンサーで検値して音の大きさが変わるというシロモノです．

いずれも楽器というにはあまりにも怪しいモノですが，その極めつけともいえるのが，JAMINATORというギターのような格好をしたモノです．フレットの部分には13コ，弦を弾く位置には3コのスイッチが並んでいます．トレモロアームまで付いています．これらを組み合わせると，リードギターのリフが鳴るのです．リフとは，ギタリストがいい気になって弾くあの音のことです．しかも，ROMでいろいろな曲が用意されており，その曲に合わせたリフが入っているのです．ですから，リッチーブラックモア風の演奏などがスグにできるわけです．ギターの部分以外に，キーボードとドラムスのスイッチもあります．最小の努力でひとりでハードロックしていい気になれるというのがウリのようです．

なんだかよくわからないシロモノではありますが，あまりにも怪しいのでつい買ってしまいました．パーカッションとベース，サイドギターのカラオケをスタートさせたら，スイッチを操ってリフを弾きます．数秒のリフを組み合わせて音楽っぽくいじっていると，なかなかその気にさせてくれます．

リフを音楽にしよう

それにしても，世の中便利になったものです．エレキギターを買って1か月，仕事の合間に練習したのでは，なかなかカッコいいリフも弾けるようになりませんでしたが，これならリズムに乗ってスイッチを押すだけです．しかし，なまじ最初からリフがあるだけに思ったように演奏できません．たしかにそれっぽくはなるのですが，音楽

ワンラインNEWS
日本ビクターはパソコン用CGソフト「トライマジン・ジュニア」を発売した．質感表現機能に優れ，日本電気のPC-9801シリーズに対応している．

にするにはまた別のテクニックが必要です．

本来は人間が弾かなくてはいけないリフが，スイッチの組み合わせで音となってしまうというのは，プロシージャやライブラリに似ています．しかし，単体でいくらプロシージャを作っても，キチンと組み合わせることができなければ音楽として，プログラムとして成立しません．

というわけで，今回からはひとつのまとまったプログラムを作ることを通して，実戦的なプログラミングの方法を紹介することにします．

ここで作るのは，データベースです．このプログラム自体も実用的に使えるデータベースですが，ほかのプログラムからデータを参照することができるようにもしましょう．つまり，一部のルーチンをほかのプログラムに組み込んで使うことができるよう，なるべく汎用に作ろうということです．そのほかにも，前回述べたようなプロシージャを使いこなすためのテクニックを使います．

データベースの仕様決定

実際にプログラミングに入る前に，ある程度の仕様を決定しておきましょう．詳細まで決定する必要はありませんが，トップダウン手法は自由度が高いだけに，いきあたりばったりでプログラミングしていくと，まとまりがつかなくなります．

ここで作るデータベースは，次のような仕様とします．

1.メニュー構成

登録，修正，削除，検索のメニューをもつこととします．削除では，削除フラグを付けるだけで，物理的には削除しません．したがって，後で復活メニューを追加して復活させることもできるでしょう．

2.ファイルの構成

インデックスとデータのファイルを持つこととします．データファイルはランダムアクセスファイルで，データ本体が入ります．インデックスファイルはキーとなるアイテムとアクティブフラグが入ります．インデックスファイルのキーの位置が，データファイルのレコード番号となるようにします．

この仕様にしたがって，いまからプログラミングしていきます．レコードやインデックスのデータ構造は，ユーザー定義型変数にすることにします．また，後からの改造を簡単にするため，データ構造はなるべく少ない箇所の変更のみで別の構造のデータベースを作れるようにしましょう．

トップダウン手法でプロシージャツリーを作る

さて，いよいよプログラミングに入りましょう．ここで，もちろんいきなりコーディングしても構わないのですが，各プロシージャで何をすべきかをはっきりさせるためにも，ここでは例によって「構造化日本語」でプログラミングしていきましょう．

1.メインプロシージャ

まず，最初はメインルーチンを考えましょう．実際のプログラムでは定数や変数の宣言がありますが，ここでは省きます．実質的に最初にすべきことは，インデックスファイルを読み込むことです．これはメニューのどの作業にも必要になるからです．次にメニューを表示し，選択キー入力待ちになります．インデックスファイルは，各メニュー内での作業ごとにセーブすることにしましょう．万が一，プログラムが途中で異常終了することがあっても，被害を最小限にすることができます．したがって，メインプロシージャは，図1のように書くことができます．

2.メニュープロシージャ

次に，メニュープロシージャを考えましょう．このプログラムには，登録，修正，削除，検索のメニューをもつことにしました．したがって，図2のように書くことができます．

このメニューは終了メニュー以外では，各メニューの処理を終えるとメニューに戻ってくるようにします．そのため，全体をDOループの中に入れてあります．メニューを画面に表示するプロシージャの後，メニュー番号入力プロシージャの戻り値によって，各処理のプロシージャを呼び出します．

だいぶ格好がついてきたような気がしてきましたね．では次に，各メニューの実際の処理内容を考えましょう．まずは，登録メニューを考えることにします．

▼図1　メインプロシージャ

```
インデックスファイル読み込みプロシージャ(オープン/クローズ)
メニュープロシージャ
```

3.登録プロシージャ

登録メニューですべきことは，まずはデータファイルのオープンです．データ入出力画面を表示したら，キーとなるデータの入力です．ここで何らかのデータが入力されると，登録プロシージャを終了します．

入力されたキーとなるデータがなければ，新規にデータを入力することにします．データを入力し終えたら，登録するか，しないか，キャンセルかを確認します．登録するならインデックスのメモリとデータファイルを更新します．登録しないならば，再入力になります．キャンセルならば，検索キーの入力待ちに戻ります．

入力されたキーとなるデータがあれば，新規には登録できませんから，そのデータを表示し，検索キーの入力待ちに戻ります．

登録プロシージャを終るときには，新たに登録したデータがあればインデックスファイルを更新します．データファイルをクローズしてこのプロシージャを終了します．

この登録プロシージャは，図3に示します．

4.修正プロシージャ

さて，ここでずいぶんと新しいプロシージャが出てきました．これらのプロシージャは，ほかの処理でも共用できる単位で分割してあります．というのも，いずれのメニューでも，キーの検索，データの表示，確認，データの書き込みなどの処理は必要だからです．

実際に，修正プロシージャを考えてみましょう．登録処理と違うのは，検索プロシージャでキーが見つかったときに主たる処理が行われることと，新規のデータを登録する必要がないのでインデックスは変更する必要がないことです．実際には，検索したキーがあったときにはそれを表示し，内容を変更してデータファイルをアップデートします．キーがなかったときには，なかった旨を表示するだけです．構造としては，登録プロシージャよりも単純です．

図4に修正プロシージャを示します．登録プロシージャと比較してみましょう．処理内容は違っていても，ほとんど同じプロシージャを使い回すことができることがわかるでしょう．

▼図2　メニュープロシージャ

```
DO
        メニュー表示プロシージャ
        SELECT CASE メニュー番号入力プロシージャ
        CASE 登録
                登録プロシージャ
        CASE 修正
                修正プロシージャ
        CASE 削除
                削除プロシージャ
        CASE 検索
                検索プロシージャ
        CASE 終了
                終了プロシージャ
                EXIT DO
        END SELECT
LOOP
```

▼図3　登録プロシージャ

```
データファイルオープン
データ入出力画面表示プロシージャ
DO UNITL キー入力プロシージャ = Quit
        IF (キー検索プロシージャ(キー) = 0) THEN
                DO
                        データ入力プロシージャ
                        登録しますか表示
                LOOP WHILE YorNorCプロシージャ = No
                IF Yes THEN
                        インデックスアペンド
                        データ書き込みプロシージャ
                END IF
        ELSE
                データ読み込みプロシージャ
                データ表示プロシージャ
        ENDIF
LOOP
IF Touch THEN
        インデックスファイル書き込みプロシージャ(オープン/クローズ)
ENDIF
データファイルクローズ
```

5.削除プロシージャ

次に，削除プロシージャを考えましょう．削除処理は，すでに登録してあるデータにのみ有効であるという点は修正処理と同じです．修正処理と違ってくるのは，キーのあるデータを修正するわけではなく削除することです．もっとも，仕様のときに決めた通り，削除はインデックスのデータのアクティブフラグをインアクティブにするだけです．

また，該当データを表示してもデータの入力処理はないので，その部分のループは必要なくなります．インデックスが変更されたら，このプロシージャを終了する前にインデックスファイルを更新する作業も必要です．

このようにして考えた削除プロシージャが図5です．登録や修正プロシージャと比較してみましょう．ここでも，ほとんどのプロシージャを共用しています．

6．検索プロシージャ

最後に，検索プロシージャを考えてみます．このプロシージャは，実に単純です．キーを検索して，あれば内容を表示し，なければない旨を表示するだけです．したがって，削除メニューから実質的な削除に関係する部分を外してしまえばいいわけです．

検索プロシージャは，図6に示します．カンタンですね．

あってほしい汎用プロシージャを考える

メニューの流れから，ひと通りのプロシージャを洗い出しました．しかし実際には，その最下層となるプロシージャが必要となることがあります．たとえば，指定座標に文字列を表示するプロシージャ，指定座標にデフォルト文字列を表示し文字列を入力するプロシージャなどがあるでしょう．このほかにも，1文字入力プロシージャ，「Y」「N」「C」だけを受け付けて対応する値を返すプロシージャ，エラー表示プロシージャ，エラー表示位置クリアプロシージャなどは，あると便利です．

これらのプロシージャは，その都度作っていってもかまいませんが，汎用性を考えて，あらかじめいくつかをボトムアップで作ってみましょう．

リスト1から6に，これらの汎用プロシージャを示します．中で使われている定数は，次の節を参照してください．

定数と変数を考える

さて，いよいよプログラミングに入りましょう．全体の構造の概要は決まっているわけですから，細かいことを考えていくことにします．みかけは細かくてもプログラミングの中では重要なポイントとなるのが，定数と変数です．

以前に述べたとおり，コード中にはあちこちに数値を埋め込むと，変更が生じたときにあちこちを修正しなくてはならなくなってしまいます．数値を直接書くのではなく，定数や変数として書いておけば，最小限の変更で複数の変更ができます．

ここで考えておくべき定数や変数について考えてみましょう．まず，レコード関係があります．インデックスのデータはメモリ上に配置されますから，いくらとってもいいというわけにはいきません．一応，最大レコード数を決めておくべきでしょう．

また，画面上に表示されるメニュー

▼図4　修正プロシージャ

```
データファイルオープン
データ入出力画面表示プロシージャ
DO UNITL キー入力プロシージャ = Quit
        IF (キー検索プロシージャ <> 0) THEN
                DO
                        データ読み込みプロシージャ
                        データ表示プロシージャ
                        データ入力プロシージャ
                        変更しますか表示
                LOOP WHILE YorNorCプロシージャ = No
                IF Yes THEN
                        データ書き込みプロシージャ
                END IF
        ELSE
                データがありません表示
        ENDIF
LOOP
データファイルクローズ
```

▼図5　削除プロシージャ

```
データファイルオープン
データ入出力画面表示プロシージャ
DO UNITL キー入力プロシージャ = Quit
        IF (キー検索プロシージャ <> 0) THEN
                データ読み込みプロシージャ
                データ表示プロシージャ
                データ入力プロシージャ
                削除しますか表示
                IF YorNorCプロシージャ = YES THEN
                        インデックスをインアクティブにする
                END IF
        ELSE
                データがありません表示
        ENDIF
LOOP
IF Touch THEN
        インデックスファイル書き込みプロシージャ(オープン/クローズ)
ENDIF
データファイルクローズ
```

▼図6　検索プロシージャ

```
データファイルオープン
データ入出力画面表示プロシージャ
DO UNITL キー入力() = Quit
        IF (キー検索プロシージャ <> 0) THEN
                データ読み込みプロシージャ
                データ表示プロシージャ
        ELSE
                データがありません表示
        ENDIF
LOOP
データファイルクローズ
```

LIST1

```
'
' エラー表示
'
SUB DispError (str AS STRING)
      LOCATE ERY, ERX
      COLOR RED
      PRINT str;
      BEEP
END SUB
```

LIST3

```
'
' エラー消去
'
SUB EraseError
      LOCATE ERY, ERX
      PRINT SPC(78 - ERX);
END SUB
```

LIST4

```
'
' 1文字入力
'
FUNCTION GetChar$
      DIM dum AS STRING

     DO
           dum = INKEY$
      LOOP WHILE dum = ""
      GetChar$ = dum
END FUNCTION
```

LIST2

```
'
' 指定座標に文字列を表示
'
SUB Display (x AS INTEGER, y AS INTEGER, c AS INTEGER, str AS STRING)
      LOCATE y, x
      COLOR c
      PRINT str;
END SUB
```

LIST5

```
'
'    指定座標にデフォルト文字列を表示し文字列を入力する
'
FUNCTION GetStr$ (x AS INTEGER, y AS INTEGER, c AS INTEGER, str AS
STRING)
      DIM dum AS STRING

      Display x, y, c, str
      LOCATE y, x
      INPUT "", dum
      IF dum = "" THEN
           dum = str
      END IF
```

LIST6

```
'
'「Y」「N」「C」だけを受け付けて対応する値を返す
'
FUNCTION YNC% (str AS STRING)
      DIM dum AS STRING

       Display YNCX, YNCY, WHITE, str + " [Y(es)/N(o)/C(ancel)] ? :"
      DO
             dum = LCASE$(GetChar$)
       LOOP UNTIL dum = "y" OR dum = "n" OR dum = "c"
       SELECT CASE dum
      CASE "y"
             YNC% = YES
      CASE "n"
             YNC% = NO
      CASE "c"
             YNC% = CANCEL
       END SELECT
END FUNCTION
```


ワンラインNEWS
米AMDはバーストモードに対応した高速EP-ROM「Am27HB010」を開発・発売。ブロック単位でデータをMPUに送るためRISC型MPUなどの性能向上に役立つ。

や入出力データ，メッセージなどの位置も，ベースを定数としておきましょう．ベースの定数を変更するだけで，全体を上下左右に移動することもカンタンです．

メッセージの内容も定数とすることができますが，これはかえって見通しの悪いプログラムになってしまうこともあります．ここでは，直接必要なときに書くことにします．

データの構造は，ユーザー定義型で宣言しましょう．QuickBASICでFIELD文を使うのは邪道です．ユーザー定義型変数をレコード変数として使えば，LSETやRSETなどをする必要もなく，レコード変数でない変数と同じように扱うことができます．

さきほどのリスト1から6のプロシージャでも，すでにある定数を仮定してプログラミングしてあります．これらの定数は，実際にリスト7のように宣言することにしましょう．

また，データベースのインデックスおよびレコードは，リスト8のような構造にしましょう．ここで，ユーザー定義型を定義するだけでは新しい変数の型ができるだけです．実際にその型の変数を使うときには，宣言をすることが必要になります．

宣言インデックスはファイルから読み込んでメモリ上で処理しますから，レコード件数分の配列をKEYSTRUCT型で宣言します．もちろん，このときのMAXRECは定数としてあらかじめ宣言しておきます．

プログラミング

ここまで準備すれば，コーディングはグっとラクになります．もっとも，

```
'
' カラー関係定数
'
CONST BLACK = 0
CONST BLUE = 1
CONST GREEN = 2
CONST CYAN = 3
CONST RED = 4
CONST MAGENTA = 5
CONST YELLOW = 6
CONST WHITE = 7
CONST REVERSE = 8
CONST BLINCK = 16
'
' 戻り値関係定数
'
CONST FALSE = 0
CONST TRUE = -1
CONST YES = 0
CONST NO = 1
CONST CANCEL = 2
'
' 座標関係定数
'
CONST ERX = 5
CONST ERY = 23
CONST YNCX = 30
CONST YNCY = 20
CONST MX = 30
CONST MY = 2
CONST NX = 10
CONST NY = 4
CONST NW = 12
'
' メニュー関係定数
'
CONST RESIST = 1
CONST MODIFY = 2
CONST DELETE = 3
CONST SEARCH = 4
CONST QUIT = 5
CONST QUITCODE = "Menu"
'
' レコード関係定数
'
CONST MAXREC = 256
'
' ファイル関係定数
'
CONST INDEXFILE = "dbindex.idx"
CONST DATAFILE = "dbdata.dat"
```



コーディングの途中で見えてくる問題もありますから，そのつど柔軟に対応するようにしましょう.

さて，このようなことに注意して作成したのが，リスト9の「db. bas」です．実際のプログラミングでは，構造化日本語だけのものに比べて多少よけいなものがついています．これは，プログラムを書く過程や動作してから初めて気付いた問題点をクリアしたためです．コメントはまったくといっていいほど入っていませんが，構造化日本語と並べてみれば，やっていることは十分わかります．この事実からも，コメントに頼ったコーディングよりも，しっかりと構造化したコメントのないプログラムの方がメンテナンスもしやすいことがわかるでしょう.

定数や変数はつい宣言するのがおっくうになって，いきなり数値を書きたくなりますが，定数にまとめられるものはまとめた方が結局は早くできあがります．たとえば，各メニューの番号を表すRESISTやMODIFYといった定数は，メニュー選択の戻り値の比較だけでなく，メニュー自体の番号の表示や各処理のタイトルにも密接に関係しています．これは，あとからメニューを追加したりメニューの番号を変更したいときにも，最小限の変更ですべての関係する処理を変更することができます.

また，入出力画面関係の座標も，すべてNXやNY，NWといった定数になっています．DispData(データの表示)，DispScreen (入出力スクリーンの表示)，EraseData(データの消去)，GetKeyCode (キーの入力)，Input-Data (データの入力) といった各プロシージャでは，すべてこの定数を元にしてあります．もし，これが定数でなく直接数値が書かれたものであれば，修正することになったら大変な手間になってしまいます．定数が実に役立つことがわかるでしょう.

また，各プロシージャの仕事の範囲は，なるべく小さくして柔軟性を持たせてあります．一般に，仕事の範囲が小さいほど，汎用に使うことができます．しかし，あまりに汎用性だけを考えて作ると，処理内容がわかりにくく，全体の見通しも悪くなります．そのプログラムの中で，「あるまとまった処理をするプロシージャ」にすることも重要になってきます．この2つの相反する条件の妥協点をうまく見つけることが，イカす構造化プログラミングのポイントです.

このプログラムを見て，そのまま入力して「わーい．動いたっ!」なんていって喜んでいるだけでは，いつまでたってもイカすプログラミングはできません．どこがイカしているかをいま一度よーーーく見直してください．そして，そのテクニックを盗んでください．それが，本当にイカすプログラミングをできるようになるためのスタートなのです.

さて，来月はいよいよ最終回です．今回のプログラムは，「とりあえず動くモノ」です．もしかしたら，まだバグが潜んでいるかもしれません．また，使いにくいところもあるでしょう．無駄なところもあります．次回は，それをシェイプアップし，さらに機能も拡張してみます.

ところで，たしかにJAMINATORは，カンタンにハードロックできます．しかし，こんなことでは，いつまでたってもちゃんとギターは弾けるようにはなれないでしょう．やはり，人間地道に努力しないといけませんね.

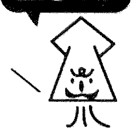



```
'
' レコード関係定数
'
CONST MAXREC = 256
'
' インデックス構造
'
TYPE KEYSTRUCT
     customer AS STRING * 20        '名前
     active AS INTEGER               'アクティブフラグ
END TYPE
'
' レコードデータ構造
'
TYPE RECSTRUCT
     customer AS STRING * 20        '名前
     address AS STRING * 60         '住所
     postnum AS STRING * 6          '〒番号
     phonenum AS STRING * 12        '電話番号
     age AS INTEGER                 '年令
     birthday AS STRING * 10        '誕生日
     blood AS STRING * 2            '血液型
     etc AS STRING * 40             'その他
END TYPE
'
' 変数宣言
'
DIM SHARED keytab(1 TO MAXREC) AS KEYSTRUCT     'キーインデックステーブル
DIM recdata AS RECSTRUCT                        'レコード変数
```

```
DECLARE SUB EraseBuffer (recdata AS ANY)
DECLARE SUB Display (x AS INTEGER, y AS INTEGER, c AS INTEGER, str AS
STRING)
DECLARE SUB DispMenu ()
DECLARE SUB QuitProc ()
DECLARE SUB ModifyProc ()
DECLARE SUB DeleteProc ()
DECLARE SUB SearchProc ()
DECLARE SUB DispScreen (m AS INTEGER)
DECLARE FUNCTION GetKeyCode% (str AS STRING)
DECLARE FUNCTION SearchKey% (str AS STRING, index AS INTEGER)
DECLARE SUB InputData (recdata AS ANY)
DECLARE SUB PutData (index AS INTEGER, recdata AS ANY)
DECLARE SUB GetData (index AS INTEGER, recdata AS ANY)
DECLARE SUB DispData (recdata AS ANY)
DECLARE SUB WriteIndex ()
DECLARE FUNCTION YNC% (str AS STRING)
DECLARE SUB ResistProc ()
DECLARE FUNCTION ReadIndex% ()
DECLARE SUB Menu ()
DECLARE SUB DispError (str AS STRING)
DECLARE SUB EraseError ()
DECLARE FUNCTION GetChar$ ()
DECLARE FUNCTION GetStr$ (x AS INTEGER, y AS INTEGER, c AS INTEGER,
str AS STRING)
DECLARE SUB EraseMessages ()
DECLARE SUB EraseData ()
'
'       データベースプログラム for QuickBASIC Version 1.0
'
'               March 5 1992
'
'           Copyright(c) 1992 Norio Sakai
'
'
' カラー関係定数
'
CONST BLACK = 0
CONST BLUE = 1
CONST GREEN = 2
CONST CYAN = 3
CONST RED = 4
CONST MAGENTA = 5
CONST YELLOW = 6
CONST WHITE = 7
CONST REVERSE = 8
CONST BLINCK = 16
'
' 戻り値関係定数
'
CONST FALSE = 0
CONST TRUE = -1
CONST YES = 0
CONST NO = 1
CONST CANCEL = 2
'
' 座標関係定数
'
CONST ERX = 5
CONST ERY = 23
CONST YNCX = 30
CONST YNCY = 20
CONST MX = 30
CONST MY = 2
CONST NX = 10
CONST NY = 4
CONST NW = 12
'
' メニュー関係定数
'
CONST RESIST = 1
CONST MODIFY = 2
CONST DELETE = 3
CONST SEARCH = 4
CONST QUIT = 5
CONST QUITCODE = "Menu"
'
' レコード関係定数
'
CONST MAXREC = 256
'
' ファイル関係定数
'
CONST INDEXFILE = "dbindex.idx"
CONST DATAFILE = "dbdata.dat"
'
' インデックス構造
'
TYPE KEYSTRUCT
    customer AS STRING * 20        '名前
    active AS INTEGER              'アクティブフラグ
END TYPE
'
' レコードデータ構造
'
TYPE RECSTRUCT
    customer AS STRING * 20        '名前
    address AS STRING * 60         '住所
    postnum AS STRING * 6          '〒番号
    phonenum AS STRING * 12        '電話番号
    age AS INTEGER                 '年令
    birthday AS STRING * 10        '誕生日
    blood AS STRING * 2            '血液型
    etc AS STRING * 40             'その他
END TYPE
'
' グローバル変数宣言
'
DIM SHARED keytab(1 TO MAXREC) AS KEYSTRUCT      'キーインデックステーブル
DIM SHARED maxindex AS INTEGER                   'インデックスの数
DIM SHARED menuname(RESIST TO QUIT) AS STRING    'メニュータイトル
menuname(RESIST) = "登録"
menuname(MODIFY) = "修正"
menuname(DELETE) = "削除"
menuname(SEARCH) = "検索"
menuname(QUIT) = "終了"
'
' メインプロシージャ
'
KEY ON
KEY 1, QUITCODE + CHR$(13)
maxindex = ReadIndex%
Menu
END
'
' 削除プロシージャ
'
SUB DeleteProc
    DIM recdata AS RECSTRUCT                     'レコード変数
    DIM inkeycode AS STRING * 20
    DIM index AS INTEGER
    DIM ret AS INTEGER
    DIM touch AS INTEGER

    touch = NO
    OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
    DispScreen DELETE
    DO UNTIL GetKeyCode%(inkeycode) = FALSE
        EraseBuffer recdata
        EraseData
        EraseError
        IF SearchKey(inkeycode, index) = TRUE THEN
            EraseMessages
            GetData index, recdata
            DispData recdata
            ret = YNC%("削除しますか")
            IF ret = YES THEN
                keytab(maxindex).active = NO
                touch = YES
            END IF
        ELSE
            DispError "データがありません。"
        END IF
        EraseMessages
    LOOP
    IF touch = YES THEN
        WriteIndex
    END IF
    CLOSE #2
END SUB
'
' データ表示
'
SUB DispData (recdata AS RECSTRUCT)
    Display NX + NW, NY + 0, WHITE, recdata.customer
    Display NX + NW, NY + 2, WHITE, recdata.address
    Display NX + NW, NY + 3, WHITE, recdata.postnum
    Display NX + NW, NY + 4, WHITE, recdata.phonenum
    Display NX + NW, NY + 5, WHITE, MID$(STR$(recdata.age), 2)
    Display NX + NW, NY + 6, WHITE, recdata.birthday
    Display NX + NW, NY + 7, WHITE, recdata.blood
    Display NX + NW, NY + 8, WHITE, recdata.etc
END SUB
'
' エラー表示
'
SUB DispError (str AS STRING)
    LOCATE ERY, ERX
    COLOR RED
    PRINT str;
    BEEP
END SUB
'
' 指定座標に文字列を表示
'
```

LIST9



db. bas

```
SUB Display (x AS INTEGER, y AS INTEGER, c AS INTEGER, str AS STRING)
    LOCATE y, x
    COLOR c
    PRINT str;
END SUB

'
' メニュー表示
'
SUB DispMenu
    CLS
    Display MX, MY, CYAN, "*** データベース (メニュー)***"
    Display MX, MY + 1 + RESIST, CYAN, STR$(RESIST) + ". 登録"
    Display MX, MY + 1 + MODIFY, CYAN, STR$(MODIFY) + ". 修正"
    Display MX, MY + 1 + DELETE, CYAN, STR$(DELETE) + ". 削除"
    Display MX, MY + 1 + SEARCH, CYAN, STR$(SEARCH) + ". 検索"
    Display MX, MY + 1 + QUIT, CYAN, STR$(QUIT) + ". 終了"
    Display MX, MY + 3 + QUIT, CYAN, "選択してください:"
END SUB

'
' データ入出力画面表示
'
SUB DispScreen (m AS INTEGER)
    CLS
    Display MX, MY, CYAN, "*** データベース (" + menuname(m) + ")***"
    Display NX, NY, CYAN, "名 前 :"
    Display NX, NY + 2, CYAN, "住 所 :"
    Display NX, NY + 3, CYAN, "郵便番号:"
    Display NX, NY + 4, CYAN, "電話番号:"
    Display NX, NY + 5, CYAN, "年 令 :"
    Display NX, NY + 6, CYAN, "誕生日 :"
    Display NX, NY + 7, CYAN, "血液型 :"
    Display NX, NY + 8, CYAN, "その他 :"
END SUB

'
' データバッファをクリアする
'
SUB EraseBuffer (recdata AS RECSTRUCT)
    recdata.customer = ""
    recdata.address = ""
    recdata.postnum = ""
    recdata.phonenum = ""
    recdata.age = 0
    recdata.birthday = ""
    recdata.blood = ""
    recdata.etc = ""
END SUB

'
' 表示しているデータをクリアする
'
SUB EraseData
    Display NX + NW, NY + 2, WHITE, SPACE$(79 - NX - NW)
    Display NX + NW, NY + 3, WHITE, SPACE$(79 - NX - NW)
    Display NX + NW, NY + 4, WHITE, SPACE$(79 - NX - NW)
    Display NX + NW, NY + 5, WHITE, SPACE$(79 - NX - NW)
    Display NX + NW, NY + 6, WHITE, SPACE$(79 - NX - NW)
    Display NX + NW, NY + 7, WHITE, SPACE$(79 - NX - NW)
    Display NX + NW, NY + 8, WHITE, SPACE$(79 - NX - NW)
END SUB

'
' エラー消去
'
SUB EraseError
    LOCATE ERY, ERX
    PRINT SPC(78 - ERX);
END SUB

'
' YNCのメッセージを消去する
'
SUB EraseMessages
    Display YNCX, YNCY, WHITE, SPACE$(79 - YNCX)
END SUB

'
' 1文字入力
'
FUNCTION GetChar$
    DIM dum AS STRING

    DO
        dum = INKEY$
    LOOP WHILE dum = ""
    GetChar$ = dum
END FUNCTION

'
' データファイルから指定データを取り出す
'
SUB GetData (index AS INTEGER, recdata AS RECSTRUCT)
    GET #2, index, recdata
END SUB

'
' キーを入力する
'
FUNCTION GetKeyCode% (str AS STRING)
    DIM i AS INTEGER

    DO
        str = RTRIM$(LTRIM$(GetStr(NX + NW, NY, WHITE, "")))
    LOOP WHILE str = ""
    IF str = QUITCODE THEN
        GetKeyCode% = FALSE
    ELSE
        GetKeyCode% = TRUE
    END IF
END FUNCTION

'
' 指定座標にデフォルト文字列を表示し文字列を入力する
'
FUNCTION GetStr$ (x AS INTEGER, y AS INTEGER, c AS INTEGER, str AS
STRING)
    DIM dum AS STRING

    Display x, y, c, str
    LOCATE y, x
    INPUT "", dum
    IF dum = "" THEN
        dum = str
    END IF
    GetStr$ = dum
END FUNCTION

'
' データを入力する
'
SUB InputData (recdata AS RECSTRUCT)
    recdata.address = GetStr(NX + NW, NY + 2, WHITE,
recdata.address)
    recdata.postnum = GetStr(NX + NW, NY + 3, WHITE,
recdata.postnum)
    recdata.phonenum = GetStr(NX + NW, NY + 4, WHITE,
recdata.phonenum)
    recdata.age = VAL(GetStr(NX + NW, NY + 5, WHITE,
MID$(STR$(recdata.age), 2)))
    recdata.birthday = GetStr(NX + NW, NY + 6, WHITE,
recdata.birthday)
    recdata.blood = GetStr(NX + NW, NY + 7, WHITE, recdata.blood)
    recdata.etc = GetStr(NX + NW, NY + 8, WHITE, recdata.etc)
END SUB

'
' メニューを表示し対応するプロシージャを呼び出す
'
SUB Menu
    DO
        DispMenu
        SELECT CASE VAL(GetChar$)
        CASE RESIST
            ResistProc
        CASE MODIFY
            ModifyProc
        CASE DELETE
            DeleteProc
        CASE SEARCH
            SearchProc
        CASE QUIT
            QuitProc
            EXIT DO
        END SELECT
    LOOP
END SUB

'
' 修正プロシージャ
'
SUB ModifyProc
    DIM recdata AS RECSTRUCT                'レコード変数
    DIM inkeycode AS STRING * 20
    DIM index AS INTEGER
    DIM ret AS INTEGER

    OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
    DispScreen MODIFY
    DO UNTIL GetKeyCode%(inkeycode) = FALSE
        EraseBuffer recdata
        EraseData
        EraseError
        IF SearchKey(inkeycode, index) = TRUE THEN
            DO
                EraseMessages
                GetData index, recdata
```

```
                DispData recdata
                InputData recdata
                ret = YNC%("変更しますか")
            LOOP WHILE ret = NO
            IF ret = YES THEN
                recdata.customer = inkeycode
                PutData index, recdata
            END IF
        ELSE
            DispError "データがありません。"
        END IF
        EraseMessages
    LOOP
    CLOSE #2
END SUB

'
'  データを指定レコードに書き出す
'
SUB PutData (index AS INTEGER, recdata AS RECSTRUCT)
    PUT #2, index, recdata
END SUB

'
' 終了処理
'
SUB QuitProc
    Display YNCX, YNCY, YELLOW, "データベースを終了します"
END SUB

'
'  インデックスファイルの読み込み
'
FUNCTION ReadIndex%
    DIM i AS INTEGER

    i = 0
    OPEN INDEXFILE FOR APPEND AS #1
    CLOSE #1
    OPEN INDEXFILE FOR INPUT AS #1
    DO WHILE NOT EOF(1) OR i >= MAXREC - 1
        i = i + 1
        INPUT #1, keytab(i).customer
        INPUT #1, keytab(i).active
    LOOP
    CLOSE #1
    ReadIndex% = i
END FUNCTION

'
' 登録プロシージャ
'
SUB ResistProc
    DIM recdata AS RECSTRUCT                'レコード変数
    DIM inkeycode AS STRING * 20
    DIM index AS INTEGER
    DIM ret AS INTEGER
    DIM touch AS INTEGER

    touch = NO
    OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
    DispScreen RESIST
    DO UNTIL GetKeyCode%(inkeycode) = FALSE
        EraseBuffer recdata
        EraseData
        EraseError
        IF SearchKey(inkeycode, index) = FALSE THEN
            DO
                EraseMessages
                InputData recdata
                ret = YNC%("登録しますか")
            LOOP WHILE ret = NO
            IF ret = YES THEN
                maxindex = maxindex + 1
                keytab(maxindex).customer = inkeycode
                keytab(maxindex).active = YES
                recdata.customer = inkeycode
                PutData maxindex, recdata
                touch = YES
            END IF
        ELSE
            GetData index, recdata
            DispData recdata
            DispError "登録済です。"
        END IF
        EraseMessages
    LOOP
    IF touch = YES THEN
        WriteIndex
    END IF
    CLOSE #2
END SUB
```

```
'
'   指定したキーに対応するインデックスを探し、あったかどうかを返す
'
FUNCTION SearchKey% (str AS STRING, index AS INTEGER)
    DIM i AS INTEGER

    FOR i = 1 TO maxindex
        IF RTRIM$(LTRIM$(str)) = RTRIM$(LTRIM$(keytab(i).customer))
AND keytab(i).active = YES THEN
            index = i
            SearchKey% = TRUE
            EXIT FUNCTION
        END IF
    NEXT i
    SearchKey% = FALSE
END FUNCTION

'
' 検索プロシージャ
'
SUB SearchProc
    DIM recdata AS RECSTRUCT                'レコード変数
    DIM inkeycode AS STRING * 20
    DIM index AS INTEGER
    DIM ret AS INTEGER

    OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
    DispScreen SEARCH
    DO UNTIL GetKeyCode%(inkeycode) = FALSE
        EraseBuffer recdata
        EraseData
        EraseError
        IF SearchKey(inkeycode, index) = TRUE THEN
            EraseMessages
            GetData index, recdata
            DispData recdata
        ELSE
            DispError "データがありません。"
        END IF
    LOOP
    CLOSE #2
END SUB

'
'  インデックスファイルを書き込み
'
SUB WriteIndex
    DIM i AS INTEGER

    OPEN INDEXFILE FOR OUTPUT AS #1
    FOR i = 1 TO maxindex
        WRITE #1, keytab(i).customer
        WRITE #1, keytab(i).active
    NEXT i
    CLOSE #1
END SUB

'
'「Y」「N」「C」だけを受け付けて対応する値を返す
'
FUNCTION YNC% (str AS STRING)
    DIM dum AS STRING

    Display YNCX, YNCY, WHITE, str + " [Y(es)/N(o)/C(ancel)] ? :"
    DO
        dum = LCASE$(GetChar$)
    LOOP UNTIL dum = "y" OR dum = "n" OR dum = "c"
    SELECT CASE dum
    CASE "y"
        YNC% = YES
    CASE "n"
        YNC% = NO
    CASE "c"
        YNC% = CANCEL
    END SELECT
END FUNCTION
```

98活用アセンブラ技法

RC-232C編 第1回

DOS BIOSの置き換え

星野 操

今回は，RS-232Cドライバの作成を試みる．ほかの多くの機能と同様に，シリアルについてもROM BIOSのサポートが存在する．これらは，PC-9801ではINT 19Hで実行され，IBM PCではINT 14Hで実行される．今月はDOS BIOSの部分のみを記述し，次回にキャラクタデバイスとして完成させる．

IBM PCのRS-232C BIOS

一般にIBM PCのBIOSは，よくできているといわれています．とくにビデオ関係など，スピードを度外視すればほとんどの操作がBIOSで可能となるようになっています．このため，たとえビデオアダプタを交換しても，BIOSでカバーできる範囲であればハードウェアの差異を気にせずにプログラムすることができます．

このような「できのいい」IBM PCのBIOSの中でも，RS-232Cに関するものは例外的に「使いものにならない」できあがりになっています．最大の問題は，これが受信割り込みをサポートしていないことにあります．

シリアル制御

シリアルの制御というと，要求される最低限の機能としては，初期化・1文字送信・1文字受信・受信状態報告，というようなものになります．通常送信機能については，割り込みではなくポーリングを使用し，送出可能になるまで待ち状態に入り，それから復帰するようになっています．送信機能は「表」のプログラムが明示的に呼び出すものであり，比較的実行の同期が取りやすいためです．

これに対して，受信の事象は「表」のプログラムとは非同期に発生します．受信についてもポーリングを使用し，つねにデータが届いたかどうかを調べるというのは現実的な手法ではありません．このような方法だと，たとえばディスクアクセスなどにいってポーリングがとぎれてしまうような場合には，送られてきた文字を取りこぼすことになってしまいます．

そのため，シリアルの受信は割り込みを使用するのが常識です．割り込みを使用しないで済むのはボーレートが比較的遅い場合，またはXMODEMのようにハンドシェークが明示的に行われ，予期しない時に受信データが到着しないようになっている場合に限られます．

PC-9801のROM BIOSは，割り込みによるデータ受信をサポートしています．しかし，IBMのそれは割り込みをサポートしていません．ポーリングによるステータスの報告，そしてデータが届いている場合の1文字直接読み込みの機能があるだけです．現実問題として，通常のモデムや対PCとヌルモデムで直接結線したような場合には，BIOSはほとんど実用にたえません．

LOWメモリとワークエリア

このような仕様になった理由としては，LOWメモリの使い方の問題があるといえます．IBM PCにしろPC-9801にしろ，メモリの下位アドレス，割り込みベクタの直後あたりにワークエリアが存在します．この領域には，たとえばキーバッファ関係のワークエリア，システム状態を示すフラグ，IBM PCであればビデオ関係の諸源(カーソル位置やカーソル形状など)などが格納されています．

PC-9801ではこの領域の詳細な情報は，あまり公開されていません．もちろん，登録開発者の元にはそれなりの資料が配布されているのでしょうが，基本的にユーザーサイドには情報が伝わってきません．これに対して，IBM PCは元来BIOS自体のリストが公開されていたため，これらのワークエリアの内容は周知となっています．

ですからプログラムが，これらのワークエリアにかかわる操作をする場合(たとえばカーソル位置をBIOSを経由せずに直接移動させる場合)には，対応するワークエリアに関しても処理を行うことがマナーとされています．

また，PC-9801には，書き込み専用で読み込みができないポートが数多くあります．アナログパレットレジスタなどはその典型で，一般に現在設定されているアナログパレットの値を知ることは不可能です．このようなハードウェアの構造であっても，特定のワークエリアをアナログパレットのために用意し，アナログパレット自体を操作した場合には，そのエリアに設定した値を格納しておく，というような処理を義務づければ問題はかなり解決したはずです．つまり，これら下位アドレスのワークエリアの構成とその情報開示が，プログラム間のいわゆる「相性」を増すために不可欠だったわけです．

しかし，これらLOWメモリのワークエリアは，プログラムとBIOSのコミュニケーションの場として設けられたわけではないでしょう．よりシンプルな，単なるワークエリア以上の意味はもっていなかったに違いありません．たとえばキー入力において，タイプアヘッドをサポートするためには，どうしても先行入力されたキーの情報をどこかに記憶していなければなりません．それを実行するプログラム自体はROMに書き込まれています．しかし，その作業領域はどこかのRAMに確保しないといけません．このための領域がLOWメモリなのです．つまりこれらのワークエリアは，あくまでBIOSが自身の機能を実現するために用意された領域なのです．

キーバッファの場合は，明らかにシステムに不可欠な機能ですし，タイプ

アヘッドバッファのサイズもそう大きなものは必要ありません．IBM PCもPC-9801もいずれも15文字程度の先行入力しか受け付けないようになっていますし，これだけでも十分に実用にたえます．

これに対して，RS-232Cの場合は必要となる情報領域の大きさが格段に違ってきます．最低128キャラクタ，多い場合には1024キャラクタ以上のバッファ領域が使用されます．BIOSの作りにもよりますが，受信データとともに受信ステータスも記録しようとすると，1キャラクタあたり2バイトあるいはそれ以上のデータ領域が必要になります．このような大きなワークエリアを，使用頻度は決して高いとはいえないシリアル受信のために割り当てておくことは，あまり現実的とはいえません．

そこでIBM PCの場合，割り込みで受信されたキャラクタを格納する領域をまったく用意せず，単にポーリングによる受信サポートのみにとどめたのです．しかし，そのために現実問題としては実用にたえないBIOSになってしまいました．IBM PCでは，およそシリアルを使うソフトウェアは（プリンタ出力など送信のみで済むような場合を除き）自前でドライバをもっているのがあたりまえであり，またこの機能を補強するために，常駐タイプの置き換えドライバがいくつも発表されています．

PC-9801の RS-232C BIOS

これに対してPC-9801はもっと賢い方法を取っています．これは割り込みによるキャラクタ受信をサポートしています．もちろん，そのためのバッファ領域をLOWメモリに取るわけにはいきません．ですからBIOSの初期化時に，「表」のプログラムがデータ領域のアドレスを渡し，そこをBIOSがバッファとして使用するようになっているのです．こうすれば，LOWメモリのワークエリアは，バッファのアドレスを記憶しておくだけで済みます．この時にバッファサイズは任意に取れるようになっています．シリアルを使用するアプリケーションの必要度合いによって，バッファサイズを可変にすることが可能となっています．

バッファには受信したキャラクタと，その時のステータスが格納されるようになっています．ですから，パリティエラーが発生したような場合には，どのキャラクタで問題が起こったかをデータ取得時に把握することができます．このBIOS機能は，N88 BASICなどが使用しています．

もう1つの RS-232C BIOS

PC-9801の場合には，このROM BIOSのほかにもう1つのRS-232C BIOS機能が存在しています．受信バッファのアドレスを指定する，というのは十分に汎用的ですがDOSで使う分には冗長に過ぎる面が否定できません．また，DOSの場合にはシリアルでエラーを把握してもそれほどメリットがありません．ですから，DOSのシリアルキャラクタデバイス，AUXを組み込んだ場合には，ROM BIOSとは別の，専用のBIOSが組み込まれることになります．

このBIOSも，前記のROM BIOSとほぼ同様の機能をもっていますが，多少異なるのは受信バッファに受信ステータスが記録されない点，また受信バッファの位置とサイズ（256キャラクタ）が固定されている点です．つまり，デバイスドライバが自前でメモリを確保することが可能になったため，このようなバッファ領域の割り当てが可能となったのです．これによって，アプリケーションが単にBIOSを呼び出す分には簡単（自前のバッファを用意する必要がない）になりましたが，その分汎用性は少なくなっています．

以前のPC-9801用MS-DOSでは，AUXは完全にシステムに組み込まれていました．しかし，3.11の比較的新しいバージョンから，AUXはRSDRV.SYSを組み込まないとインストールされないようになってしまいました．このため，現在ではこのMS-DOSのドライバがインストールされていることを無条件に仮定することはできません．

かなり古いバージョンのMS-DOSでは，MS-DOSのドライバがROM BIOSのドライバを覆い隠して，ROM BIOSのそれはまったく使用できなかったように記憶しています．しかし，現在のドライバはより汎用性をもっています．MS-DOSのBIOSを利用する場合には，DSレジスタの値を60Hに設定することが義務づけられています．BIOS呼び出し時のDSレジスタの値をチェックし，これが60Hである場合にはMS-DOSのBIOS処理を，そうでない場合にはROM BIOS処理を呼び出すようになっています．そこで，ROM BIOSを使用することを前提としてプログラムを書いておけば，RSDRV.SYSの有無にかかわらずプログラムを動作させることが可能です．汎用性を考えると，可能ならばROM BIOSを使用するようにしておいたほうが無難といえるでしょう．

リスト1　ASYNC.ASM

```
        include condx.inc
;-------------------------
; tx/rx buffer definition
;

; buffer size contstans
RX_BUF_SIZE     equ     256
TX_BUF_SIZE     equ     256

; actual buffer
rx_buffer       db      RX_BUF_SIZE dup (?)
tx_buffer       db      TX_BUF_SIZE dup (?)

; pointer of buffer
rx_top          dw      rx_buffer
rx_tail         dw      rx_buffer
tx_top          dw      tx_buffer
tx_tail         dw      tx_buffer
tx_int_active   dw      0


;-------------------------
; flow control definition
;

FLOW_NONE       equ     0
FLOW_HARD       equ     1
FLOW_SOFT       equ     2

flow_type       dw      FLOW_NONE
tx_flowed_off   db      0
rx_flowed_off   db      0
flow_char_required      db      0
flow_char_code  db      0

XON_CODE        equ     11h
XOFF_CODE       equ     13h
RSON_COMMAND    equ     37h
RSOFF_COMMAND   equ     17h


;-------------------------
; hardware definition
;

DATA_PORT       equ     30h
COMMAND_PORT    equ     32h
INT_PORT        equ     0
MASK_PORT       equ     2
INT_MASK        equ     35h
TIMER_CONTROL   equ     77h
TIMER_COUNT     equ     75h


;-------------------------
; interrupt definition
;

INT_NUMBER      equ     4
INT_VECTOR      equ     (8+INT_NUMBER)*4
INT_BIT         equ     10h
TXINT_ACTIVE_MASK       equ     5
TXINT_INACTIVE_MASK     equ     1

;-------------------------
; interrupt interface
;

int_work        struc
int_num         dw      ?
int_func        dw      ?
int_old         dd      ?
int_work        ends

interrupt_list  label   byte
i0c     int_work        <0ch,dummy_int,0>
i0f     int_work        <0fh,dummy2_int,0>
i19     int_work        <19h,bios_entry,0>
        dw      0

dummy_int:
        push    ax
        mov     al,20h
        out     INT_PORT,al
        pop     ax
dummy2_int:
        iret


;-------------------------
; output interrupt trigger macro
; input:
;       al: data to output

output_trigger  macro
        ; port output
        out     DATA_PORT,al

        ; setup tx interrupt
        mov     tx_int_active,1
        mov     al,TXINT_ACTIVE_MASK
        out     INT_MASK,al
        endm


;-------------------------
; tx advance pointer macro
; argument:
;       reg: pointer

tx_advance      macro   reg
        inc     reg
        cmp     reg,offset tx_buffer+TX_BUF_SIZE
        @if     ae
                mov     reg,offset tx_buffer
        @endif
        endm


;-------------------------
; rx advance pointer macro
; argument:
;       reg: pointer

rx_advance      macro   reg
        inc     reg
        cmp     reg,offset rx_buffer+RX_BUF_SIZE
        @if     ae
                mov     reg,offset rx_buffer
        @endif
        endm


;-------------------------
; interrupt support routine
; broken:
;       ax,bx

rxintr  proc    near
        ; data read
        in      al,DATA_PORT

        ; check flow control char
        cmp     flow_type,FLOW_SOFT
        @if     z
                ; check xoff
                cmp     al,XOFF_CODE
                @if     z
                        ; flow off
                        mov     tx_flowed_off,1
                        pop     bx
                        ret
                @endif

                ; check xon
                cmp     al,XON_CODE
                @if     z
                        ; flow on
                        cmp     tx_flowed_off,0
                        @if     nz
                                ; output char if need
                                cmp     tx_int_active,0
                                @if     z
                                        ; first check flow character
                                        mov     bx,tx_top
                                        cmp     bx,tx_tail
                                        @if     z
                                                ; no data to output, shortcut
                                                ret
                                        @endif

                                        ; take data
                                        mov     al,cs:[bx]

                                        ; advance pointer and save
                                        tx_advance <bx>
                                        mov     tx_top,bx

                                        ; output data
                                        output_trigger
                                @endif
                        @endif

                        mov     tx_flowed_off,0
                        ret
                @endif
        @endif

        ; store it into buffer
        mov     bx,rx_tail
        mov     cs:[bx],al

        ; advance pointer
        rx_advance <bx>

        cmp     bx,rx_top
        @if     nz
                ; more buffer space
                mov     rx_tail,bx

                ; check flow
                sub     bx,rx_top
                @if     c
                        ; adjust
                        add     bx,RX_BUF_SIZE
                @endif

                @mc
                        cmp     bx,(RX_BUF_SIZE*2)/3
                        @cond   ae
                        cmp     rx_flowed_off,0
                        @cond   z
                        cmp     flow_type,FLOW_NONE
                        @cond   nz
                @if
                        ; flow off control require
                        mov     rx_flowed_off,1
                        cmp     flow_type,FLOW_HARD
                        @if     z
```

高速送・受信における問題

さて一般的な構成でのPCにおいて，もっとも高速のサービスを必要とする割り込みは，シリアルでしょう．しかし，すでに割り込みでの応答は限界に近づいているといえます．日本ではまだモデムというと，2400bpsが標準的です．しかし，U.S.A.ではV.32bisモデム（14400bps）が非常に安価になってきています．

9600bpsでの非同期受信の割り込み間隔は，約1msです．このため，いろいろなハードウェア割り込み，またソフトウェアの割り込み禁止領域ではこの程度の時間の割り込み禁止は許容されました．しかし，より高速のシリアル受信をサポートしようとすると，許容できる割り込み禁止時間はどんどん短くなります．38400bpsでは250μs程度となってしまいます．これでは，文字を取りこぼすことなくデータを受信することは，非常に困難です．ましてやV.32bisにV.42bis圧縮をサポートすると，公称データ転送レートは57.6Kbpsにもなってしまいます．

このため，IBM PCではより高性能なシリアルチップ（16550）を使用することが徐々に開始されつつあります．以前から使用されていたシリアルチップ（16450）と互換性を有し，特定の操作をすることによって内部FIFOバッファが有効となり，16キャラクタ程度を内部に蓄えることができるという石です．これを使用することによって，割り込み応答速度が遅くなっても（定常的ではなく，主として散発的に）文字を取りこぼさないで済む確率が非常に高くなります．この石は16450にピンコンパチブルであるので，ソケットを使用しているボードであればチップを差し替えるだけで高速モデムに対応できるようになります．もっとも，最近のインターフェースボードでは，シリアル・パラレルのインターフェースがみな1チップに内蔵されているカスタムチップが使われているので，これと置き換えることは困難です．

PC-9801の問題点

これに対して，PC-9801のシリアルはスピードの高速化に対しては非常に貧弱です．IBM PCのように，シリアルチップのサポートがないのはもちろんのこと，使用可能な最高ボーレートがかなり低くなってしまうのです．

PC-9801には，いわゆる5M系列と8M系列があります．5M系列は文字どおり5M／10M／20Mなどのシステムクロックを有するものであり，8M系列は8M／16Mのクロックを有するものです．本来システムクロックとシリアル（タイマICの入力クロック）が同期している必要はまったくないのですが，PC-9801の昔の設計ではクリスタルチップをけちったのか，システムクロックを変更するとタイマICの入力クロックが変更されます．それが現在まで踏襲されて，2つのクロック系列ができてしまったのです．

5M系列では，（シリアルチップのスペックはオーバーすることになるでしょうが）シリアルのクロックを38400bpsまで上げることが可能です．これに対して8M系列の場合は，分周比の関係から9600bpsまでしかサポートすることができません．V.32の9600bpsモデムを接続するにも不十分なクロックしか得られないのです．このことから，PC-9801と高速モデムの接続では，将来非常に多くの問題が発生することになるでしょう．

さらに悪いことに，NESAバスシステムは，8M系列であるという話があります．H98Sは486/SXの20MHzのチップを使用していますが，シリアル関係を支配するクロックは8M系列になっているということです．CPUクロック12MHzというシステムが登場して以来，CPUクロックとタイマクロックの独立性は明らかになっていますが，せっかくCPUクロックが10M系列であるのに，タイマが8M系列であるというのは解せません．NESAシステムバスの関係であるという話もあり，これがほんとうだとするとNESAバスも罪作りです．

V.32bis／V.42bisの高速モデムになると，こんどは57.6Kbpsでターミナルと接続することが推奨されます．IBM PCではこれが可能となっているのですが，PC-9801では10M系列においても，このクロックをサポートすることはできません．高速シリアルについては，いろいろと問題が出てくることになるでしょう……．

AUXデバイスの置き換え

以上のように将来の暗いPC-9801のシリアルですが，ともかくもここではDOSのAUXデバイスを置き換えることを考えます．もちろん，DOS BIOSについても置き換えをしなければいけません．今回は，DOS BIOSの部分のみを記述し，次回にキャラクタデバイスとして仕上げることにします．

MS-DOSやROM BIOSのRSドライバには，次のような改良を要する点があります．

ワンラインNEWS
日本電信電話はUNIXの国際標準化団体であるUIの賛助会員として加盟した．同社はすでにもう1つの標準化団体であるOSFにも加盟していた．

```
                                ; hardware flow, off RS
                                mov     al,RSOFF_COMMAND
                                out     COMMAND_PORT,al
                        @else
                                ; software flow
                                cmp     tx_int_active,0
                                @if     nz
                                        ; already interrupt active
                                        mov     flow_char_required,1
                                        mov     flow_char_code,XOFF_CODE
                                @else
                                        ; directly output
                                        mov     al,XOFF_CODE
                                        output_trigger
                                @endif
                        @endif
                @endif
        @endif

        ret
rxintr  endp

;--------------------------
; tx interrupt support routine
; broken:
;       ax, bx

txintr  proc
        ; first, check flow character
        cmp     flow_char_required,0
        @if     nz
                ; output flow character
                mov     al,flow_char_code
                out     DATA_PORT,al

                ; and immidiately return
                ret
        @endif

        ; check flow control off or buffer empty
        @mc
                cmp     tx_flowed_off,0
                @cond   z
                mov     bx,tx_top
                cmp     bx,tx_tail
                @cond   nz
        @if
                ; take first data from buffer, and output
                mov     al,cs:[bx]
                out     DATA_PORT,al

                ; advance pointer
                tx_advance <bx>
                mov     tx_top,bx
        @else
                ; no data active or flowed off, disable interrupt
                mov     tx_int_active,0
                mov     al,TXINT_INACTIVE_MASK
                out     INT_MASK,al
        @endif

        ; return
        ret
txintr  endp

;--------------------------
; interrupt main routine
;

hard_int:
        ; for insurance
        cli

        ; save registers
        push    ax
        push    bx

        @begin
                ; get status flag
                in      al,COMMAND_PORT

                ; check rx ready
                test    al,2
                @if     nz
                        ; rx service
                        call    rxintr
                        @continue
                @endif

                ; check tx ready, and interrupt service flag
                @mc
                        test    al,1
                        @cond   nz
                        cmp     tx_int_active,0
                        @cond   nz
                @if
                        ; tx service
                        call    txintr
                        @continue
                @endif

                ; no requirement to service
                @break
        @again

        ; send EOI
        mov     al,20h
        out     INT_PORT,al

        ; return
        pop     bx
        pop     ax
        iret

;==========================
; BIOS support routine
;==========================

waste_time      proc
        jmp     $+2
        jmp     $+2
        jmp     $+2
        jmp     $+2
        jmp     $+2
        ret
waste_time      endp

;--------------------------
; initialize routine
; input:
;       60:68 76543210
;             |||||||x  xflow on(1), off(0)
;             ||||||X   hard flow on(1), off(0)
;             ||||xx    7bits(10), 8bits(11)
;             |||x      parity(1), no parity(0)
;             ||x       even parity(1), odd parity(0)
;             xx        1 stop bit(01), 1.5 stop bit(10), 2 stop bits(11)
;--------------------------
;       60:69 76543210
;             ooooxxxx  75b(0001), 300b(0011)..., 38400(1010) if available
;                       2nd port not supported

bios_init       proc
        push    cx

        ; ds register is 60h
        ; disable async interrupt
        in      al,MASK_PORT
        or      al,INT_BIT
        call    waste_time
        out     MASK_PORT,al

        ; initialize async
        mov     al,0
        out     COMMAND_PORT,al
        call    waste_time
        out     COMMAND_PORT,al
        call    waste_time
        out     COMMAND_PORT,al
        call    waste_time
        out     COMMAND_PORT,al
        call    waste_time
        mov     al,40h
        out     COMMAND_PORT,al
        call    waste_time

        ; get async command data
        mov     al,ds:[68h]
        mov     ah,al
        and     al,0fch
        or      al,2                    ; x16 mode only
        out     COMMAND_PORT,al
        call    waste_time
        mov     al,RSON_COMMAND
        out     COMMAND_PORT,al

        ; check system clock
        mov     bx,3328
        in      al,42h
        test    al,20h
        @if     z
                mov     bx,4096
        @endif

        ; calc clock divide rate
        mov     cl,ds:[69h]
        and     cl,0fh
        shr     bx,cl

        ; set clock
        mov     al,0b6h
        out     TIMER_CONTROL,al
        call    waste_time
        mov     al,bl
        out     TIMER_COUNT,al
        call    waste_time
        mov     al,bh
        out     TIMER_COUNT,al

        ; interrupt port set
        mov     al,TXINT_INACTIVE_MASK
        out     INT_MASK,al

        ; flow mode set
        mov     al,ah
        and     al,3                    ; check flow mode
        cmp     al,3
        @if     z
                mov     al,0            ; invalid, no flow
        @endif
        mov     ah,0
        mov     flow_type,ax

        ; work area clear
        mov     ax,offset rx_buffer
        mov     rx_top,ax
        mov     rx_tail,ax
```

①送信割り込みがサポートされていない.

出力に関しては，前記のようにポーリングでもある程度のパフォーマンスを確保することはできます．しかし，送信割り込みのサポートがあったほうがパフォーマンス的に有利であることは明らかです．とくに，ディスクアクセスとある程度の平行動作が可能となるので，ディスクデータをRS-232Cに出力する場合，またはその逆の場合など，速度的に有利になる可能性があります．

②フロー制御が単方向しかサポートされていない.

XON／OFFフローを使用することができるようになっていますが，これは受信側のそれのみに有効なものであり，XON／XOFFコードを「受信」したときに送信を再開／中断する機能は備わっていません．これを実現するためには，遅延なく受信データを引き取り，それを解釈する必要があり，これはかなり困難な作業です．とくに送信割り込みを使用する場合には，そちらとの協調動作が不可欠となります．

③ハードウェアフローがサポートされていない.

高速モデムなどとのフロー制御を実行する場合に，データの透過性なども考えるとXフローはあまり好ましいものではありません．一般的にはRS／CSのハードウェアフローが使用されることになるでしょう．CS (CTS) はシリアルチップのデータ送出をハードウェア的に抑制するので，ソフトウェアのサポートがなくても送信側のフロー制御は問題ありません．しかし受信側のフロー制御（受信バッファがいっぱいになった場合にRSを落とす作業）はドライバソフトウェアが受け持つ必要があります．

今回は，これらの点について改良を加えてみることにします．なお，RSドライバでは拡張RS-232Cもサポートしているものがありますが，今回は本体内蔵のシリアルのみをサポートすることにします．

今回のプログラム ASYNC.ASM

ではソースの方を見てみることにしましょう．

ASYNC. ASMが今回のプログラムの本体です．常駐プログラムにもデバイスドライバにもできるように汎用的な記述にしてあります．そのため，ほかのファイルでセグメントなどをセットアップして使えるように，今回の部分ではセグメント指定などは何も行っていません．また，UTILS. INCとfuncマクロは使用せずに，関数はすべてproc疑似命令を使用しています．呼び出しにも，call命令が必要となります．

送受信バッファ

さて，プログラムの最初には，送受信のバッファの指定があります．ここでは双方とも255キャラクタキューイングできるようにしてあります．受信バッファがあふれた場合には，新しいキャラクタは捨てられることになり，送信バッファがあふれそうになると，キューイングは動作せずバッファに空きができるまで待つようになります．ですから，場合によってはバッファサイズを適切に再定義する必要が生じるかもしれません．

バッファはリングバッファ構造を使用するので，データをプッシュするポイントとポップするポイントそれぞれのポインタが必要となります．それらがその次に記述されています．このほかに，現在有効なデータ長を別のカウンタに持たせる形式もありますが，今回はポインタのみでバッファを表現しています．そのため，バッファが空の状態と区別できないので，バッファが完全にフルの状態，つまりここで256キャラクタが格納されている状態は存在してはいけないことにします．IBM PCのキーボードBIOSはここと同様な処理をしているので，16キャラクタのリングバッファをもっていますが，実際に格納可能なのは15キャラクタまで，PC-9801の場合は別にカウンタをもっているので16キャラクタまるまる保持することができます．

tx_int_active

その次の，tx_int_activeは，現在送信割り込みが有効であるかどうかを示すフラグです．新たなキャラクタ送信要求があった場合に，現在割り込み駆動が有効である場合には，単にバッファの後端にデータを追加するだけで済みます．しかし，すでに送出バッファがクリアされている状況では，割り込みは停止されています．その場合には直接ポートにデータを出力し，割り込みを許可するというような処理が別に必要になります．この状況判定のためにフラグを使用します．

フローコントロール

それから，フローコントロール関係の定義があります．前記のように，今回のプログラムではハードウェアフローコントロールもサポートします．ハ

```
        mov     ax,offset tx_buffer
        mov     tx_top,ax
        mov     tx_tail,ax
        mov     ax,0
        mov     tx_int_active,ax
        mov     tx_flowed_off,al
        mov     rx_flowed_off,al
        mov     flow_char_required,al
        mov     flow_char_code,al
        mov     flow_char_required,al

        ; interrupt vector set
        push    ds
        mov     ax,0
        mov     ds,ax
        mov     word ptr ds:[INT_VECTOR],offset hard_int
        mov     word ptr ds:[INT_VECTOR+2],cs
        pop     ds

        ; enable mask
        in      al,MASK_PORT
        and     al,not INT_BIT
        call    waste_time
        out     MASK_PORT,al
        mov     ah,0

        pop     cx
        ret
bios_init       endp


;------------------------------
; set data into tx buffer
; waste_time if tx buffer is full
; input:                zz
;       cl: data

bios_set_data   proc
        ; check tx interrupt
        cli
        mov     al,cl

        @mc
                cmp     tx_int_active,0
                @cond   z
                cmp     tx_flowed_off,0
                @cond   z
        @if
                ; directly output
                output_trigger

                ; shortcut return
                sti
                ret
        @endif
        sti

        ; check buffer space
        @begin
                sti

                mov     bx,tx_tail
                tx_advance <bx>
                cmp     bx,tx_top
        @until  nz

        ; set data
        cli
        mov     bx,tx_tail
        mov     cs:[bx],al

        ; update pointer
        tx_advance <bx>
        mov     tx_tail,bx

        sti
        mov     ah,0
        ret
bios_set_data   endp


;------------------------------
; data get sub function
; return:
;       al: data
;       zflag: set if buffer empty

get_data_sub    proc
        ; disable interrupt
        cli

        ; check buffer
        mov     bx,rx_top
        cmp     bx,rx_tail
        @if     nz
                ; get and save data/flag
                mov     al,cs:[bx]
                push    ax
                pushf

                ; advance pointer
                rx_advance <bx>
                mov     rx_top,bx

                ; check flow
                @mc
                        cmp     rx_flowed_off,0
                        @cond   nz

                        ; no need to check flow type
                        mov     ax,rx_tail
                        sub     ax,bx
                        @if     c
                                add     ax,RX_BUF_SIZE
                        @endif
                        cmp     ax,RX_BUF_SIZE/3
                        @cond   b
                @if
                        ; now flow control on
                        cmp     flow_type,FLOW_HARD
                        @if     z
                                ; hard flow, set RS on
                                mov     al,RSON_COMMAND
                                out     COMMAND_PORT,al
                        @else
                                ; software flow
                                cmp     tx_int_active,0
                                @if     nz
                                        ; set data
                                        mov     flow_char_required,1
                                        mov     flow_char_code,XON_CODE
                                @else
                                        ; directly output
                                        mov     tx_int_active,1
                                        mov     al,TXINT_ACTIVE_MASK
                                        out     INT_MASK,al
                                @endif
                        @endif

                        mov     rx_flowed_off,0
                @endif

                ; resume flag and data
                popf
                pop     ax
        @endif

        sti
        mov     ah,0
        ret
get_data_sub    endp


bios_get_data   proc
        @begin
                call    get_data_sub
        @until  nz
        ret
bios_get_data   endp


;------------------------------
; get status
;

bios_get_status proc
        in      al,COMMAND_PORT
        mov     cl,al
        mov     ah,0
        ret
bios_get_status endp


;------------------------------
; get data counts
;

bios_get_count  proc
        mov     cx,rx_tail
        sub     cx,rx_top
        @if     b
                add     cx,RX_BUF_SIZE
        @endif

        mov     ah,0
        ret
bios_get_count  endp


;------------------------------
; break control
;

bios_set_break  proc
        mov     al,3fh
        out     COMMAND_PORT,al
        ret
bios_set_break  endp


bios_reset_break        proc
        mov     al,37h
        out     COMMAND_PORT,al
        ret
bios_reset_break        endp


;============================
; service routine
;

entry_array     label   word
        dw      bios_init               ; code 0
        dw      bios_set_data           ; code 1
        dw      bios_get_data           ; code 2
        dw      bios_get_status         ; code 3
        dw      bios_get_count          ; code 4
        dw      bios_set_break          ; code 5
        dw      bios_reset_break        ; code 6
entry_array_end label   word
```

```

bios_entry:
        ; first, check ds value
        push    ax
        mov     ax,ds
        cmp     ax,60h
        pop     ax
        @if     nz
                ; call rom bios
                jmp     i19.int_old
        @endif

        ; check command range
        cmp     ah,(offset entry_array_end-offset entry_array)/2
        @if     b
                ; calculate entry address
                push    bx
                mov     bl,ah
                mov     bh,0
                add     bx,bx
                call    entry_array[bx]
                pop     bx
        @endif

        ; terminate
        iret

```

ードウェアフローはRS／CSのハンドシェークのみとし，現実問題としては受信バッファの状況に応じてRSの制御だけを行い，CSの制御はチップのハードウェア機構に依存することにします．

フローに関しては，その制御タイプと，送受信双方のフローコントロール状況を記憶する必要があります．ただし，送信側のフローコントロールフラグが立つのは，ソフトウェア（Xフロー）が使用される場合のみです．

flow_char_required／flow_char_code

その次のflow_char_requiredとflow_char_codeというのは，Xフロー使用時のフローコントロールに使用します．フローコントロールキャラクタは，なるべく速やかに送出される必要があります．つまり，前記のバッファ構造だと通常のキャラクタ送出の場合のように，バッファ後端にデータが追加されるのは好ましくありません．バッファの先端に追加する必要があります．しかし，バッファがフルになっている状況も考えられないではありません．ですからフローキャラクタ専用のバッファ（ちょうどCのungetchのためのような）を設けておくのです．このようにしておけば，なんらかの理由でXOFFコードを送出しないうちにXONコードを送出する必要が生じた場合に，無駄なXOFFコードを送出する手間が省けます．

その次のXON_CODEとXOFF_CODEはそれぞれのアスキーコード，RSON_COMMAND／RSOFF_COMMANDはRSをコントロールするために，チップに送出するコマンドコードです．

ハードウェア関連定義

それからハードウェア関係の定義があります．DATA_PORT／COMMAND_PORTはシリアルチップのポート，INT_PORT／MASK_PORTはPICのポート，TIMER_CONTROL／TIMER_COUNTはボーレートジェネレータPITのポートアドレスです．INT_MASKは，送受信割り込みの割り込み諸元を決定するポートで，送信割り込みの可否をここで制御します．

次に，割り込み関係の定義があります．シリアルチップの割り込み番号とそのベクタ，PICの割り込みマスクビットの定義があります．TXINT_ACTIVE_MASK／TXINT_INACTIVE_MASKは，INT_MASKポートに出力する，送信割り込み許可・禁止のビットパターンです．

その次に，割り込みチェーンのための記述があります．DOSのRSDRV.SYSと同様な機能を持たせるために，BIOS呼び出し時にDSセグメントの値を調べ，それが60Hでない時にはROM BIOSにチェーンするようにしなければいけません．このために，前のベクタアドレスを記憶する必要があります．ハードウェア割り込みのベクタアドレスも同様に保存しておきます．dummy_intは，最初にハードウェ

ア割り込みに設定されるべきアドレスで，単にEOIコマンドを送出し復帰するだけです．

マクロ定義

それからいくつかのマクロ定義があります．output_triggerは，現在送信割り込みが禁止状態になっている場合に，まずALレジスタのデータを直接送出し，その後に送信割り込みを許可するようにします．こうして，次のデータから割り込み駆動で送出が可能になります．その次のtx_advance／rx_advanceは，それぞれTX／RXバッファポインタを進めるためのマクロです．リングバッファ構造を取っているために，ラップアラウンド処理を行わないといけません．このために使用されるマクロです．実際に調整するレジスタは，引数に取って示すことにしますが，ここではBXレジスタが使用されます．

割り込み処理ルーチン

さて，その次に送受信割り込みの処理ルーチンがあります．rxintrは受信割り込み要求を処理する部分です．ここではDOSのそれと同様に，エラーなどのステータスチェックは行いません．まず受信データを取得した後，Xフローチェックを行います．フローコントロールがソフトモードになっている場合には，XON／XOFFコードであるかどうかを調べ，それぞれに対応した処理をします．XOFFコードであれば，フロー制御中のフラグを立てます．次の送信割り込み処理時に，このフラグによって割り込み送信を中断するようにします．

逆にフロー制御中のXONコードであった場合には，当該フラグをクリアし，送出待ちデータがあった場合にはそれを送出します．このために，output_triggerマクロを使用します．

それ以外の通常のキャラクタ受信であった場合には，バッファの後端にデータを追加します．そして，受信フローが効いていず，かつバッファの2／3以上のデータがすでにある場合には，フロー制御を行います．ハードフローの場合には，単にRSを降ろすだけですが，ソフトウェアフローの場合には割り込みが駆動中であるかどうかによって，処理を変えます．駆動中であれば，前記のワークエリアにXOFFコードを設定し，そうでなければoutput_triggerを利用して割り込み駆動を開始します．

送信割り込み要求があった場合には，まずフローキャラクタ送信要求を調べます．この要求がなかった場合には，フローoff状態ではなく，かつバッファにデータが存在する場合にそれを出力します．そうでなければ，割り込み駆動を中断する必要があるので，tx_int_activeフラグをクリアし，割り込みをマスクします．

送信割り込み要求をインアクティブにする方法は2つあります．1つは今回利用したように，割り込み諸元選択のポートを利用して，直接送信割り込みを無効にする方法です．このほかに，シリアルチップの送信をインアクティブ状態にすることで，割り込み要求が発生しないようにすることもできます．しかし，今回はより簡単な方法を取っています．ここで，なにも処理をしないといつまでたっても送信割り込み要求がクリアされず，割り込みが受け付けられなくなります．

hard_int

hard_intは，これら割り込み要求をマネジメントする割り込みエントリです．今回使用する割り込み諸元は送信・受信と2つありますが，マルチプレクスされた割り込みが発生するので，対応する処理を選択して行わなければいけません．しかも，割り込み処理中に，他方の割り込み要求が発生する可能性もあります．ですから，割り込み処理ルーチンは割り込み諸元がすべてクリアされるまで，ループを回さなければなりません．割り込み要求がクリアされたことを保証しないと，エッジトリガーの割り込みは以後受け付けられなくなります．レベルトリガー割り込みになっていれば，なんら問題はなかったのですが．

このモジュールでは，データ領域はすべてCSレジスタ経由でアクセスするので，DSレジスタのセットアップは行いません．ループの中では，まずステータスレジスタを読み込み，受信割り込み要求があれば，それを処理します．そうではなくて送信割り込み要求があり，かつ割り込みがアクティブ状態にある時に送信割り込みの処理を行います．チップの送信要求フラグは立ちっぱなしになるので，同時にこのフラグも見て判定しなければいけません．チップを送信禁止状態にする方法をとっても，チップのフラグはクリアされないので，やはり割り込み状態を示すフラグを管理する必要があることには変わりがありません．

すべての割り込み要求がクリアされた場合には，EOIを送出して終了します．

BIOS処理

それからBIOS機能の処理部分となります．まず，初期化ファンクションの処理です．オリジナルのBIOSは当然のことながらハードフローの制御，また19200bps以上のボーレート設定の機能はありません．これらの設定状況はセグメント60Hの特定のアドレスにあらかじめ書き込んでおくことになるのですが，ここでは，ハードフローのために空きビットを1つ使用し，ボーレートについてはそのまま拡張していくことにしました．

処理は，まず最初にシリアルの割り込みを禁止します．これはPICのマスクポートを設定することで行います．それからシリアルチップを初期化し，モード設定を行います．このために，前記のワークエリアを参照します．次にボーレートジェネレータの分周比を設定します．8M系列と5M系列では基準値が異なるのですが，それらをI/Oポートを参照することで判断します．これはLOWメモリの特定のビットを見ることでも処理できます．それから，ボーレートに応じて基準分周比をシフトすることで実際の分周比を算出します．ここで8M系列で9600bps以上の要求があった場合には，もちろん正確な値は設定不能です．

次に，フロー制御モードを設定します．ここで拡張されたビットが使用されます．最後にワークエリアを初期化し，割り込みベクタを設定した後に，再び割り込みを許可し終了します．

データ送信ファンクション

次に，データ送信ファンクションですが，フロー制御が効いておらず，かつ割り込み駆動状態でない場合には，output_triggerを利用して割り込みを駆動します．そうでない場合には，バッファの終端にデータを追加しますが，もし空き領域がない場合には空きができるまで送出を待ちます．ここで，ワークエリアを触る時には，割り込み禁止状態にします．最後にポインタをアップデートしてから終了します．

データ受信ファンクション

データ受信ファンクションは，get_data_subという下請けファンクションを利用します．これは受信バッファにデータがあればそれを取得し，なければZフラグをセットして終了するものです．ここでデータを取得した場合に，フロー解除する必要が発生した場合には，それに対応した処理をします．ひんぱんなフロー制御を避けるために，フロー制御を解除するのは，残データがバッファの1/2以下を基準とします．ハードフローの場合はRSをクリアするわけですが，ソフトフローの場合には受信割り込み時の処理と同様な処理をすることになります．実際のBIOSファンクションでは，データが用意されるまで待つことになるので，Zフラグがクリアされるまで呼び出しを繰り返します．

このほかにシリアルチップのステータスを取得するファンクション，ブレーク処理ファンクションなどが必要です．受信バッファの長さを取得するファンクションはラップアラウンド処理をすることになります．

BIOSエントリ

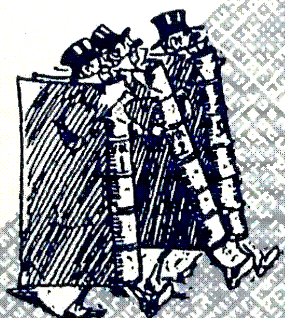
最後に，BIOSエントリの記述があります．BIOSファンクションはテーブル参照で呼び出すことにするため，そのためのテーブルが用意されています．エントリポイントでは，まずDSの値をチェックし，これが60Hでない場合には，ROM BIOS(あるいは，それ以前に設定されていたエントリポイント)を呼び出します．そうでなければ，テーブル経由で各ファンクションを呼び出すことになります．

参考文献

1) テクニカルデータブック／アスキー出版
2) PC-98用拡張システムコール詳説／インターフェース4月号別冊付録 CQ出版

IC千夜一夜

第11回——シリアルインターフェース用LSI



虎岩 登

最近は，パソコン通信とやらもかなり普及したようですね．本誌でも，編集部に専用のコンピュータを置いてサービスしているようですが．さて，その際，電話回線にコンピュータを接続するのに，モデムという装置を使うのが一般的です．今回は，直接モデムに関連はしないのですが，コンピュータとモデムを接続するときに使うことの多い，RS-232Cと呼ばれているインターフェース用のIC，シリアル(直列)インターフェース用LSIを取り上げることにします．

直列転送方式の変遷

モールス符号から調歩同期式へ

RS-232C(現在では規格が改訂されて，正式にはEIA-232Dと呼ぶべきもののようです)の特徴はデータを1bitずつ，時分割して転送するという点にあります．まず，この直列転送の歴史的な変遷を振り返ってみましょう．

通信の歴史というのを眺めると，昔は「のろし」を使うとか，「伝令」が走って伝えたとかいろいろありますが，近代的な電気通信となるとモールス符号による電信通信が最初のものでしょう．モールス符号は，長点と短点の組み合わせで文字を表します．短点の長さを1とすると，長点の長さは3になります．また，1文字の間の点と点の間は1だけ空け，文字と文字の間は3だけ空けます．同様に単語と単語の間は7空けます．そうして，アルファベットや数字，いくつかの区切り符合をを表現します．よく使うアルファベットは短く送れるような符号が割り当てられています．たとえば，Eは・でTは－になり，Qは－－・－になります．このような符号系を用いれば，電波なら単一周波数の電波の断続だけで，有線なら1対をなす2本の電線だけで通信が可能になります．音声を電気信号に変えて電送する技術が生まれる前，あるいはそれが存在しても高価な技術だったころ，盛んに使われました．いまではアマチュア無線の一部が使うだけといっても過言ではない状況になっています．

モールス符号は文章を電送するにはたいへん役に立ちましたが，これを機械で自動的に処理できないかという発

想が当然生まれてきます．機械電信の誕生です．このために作られた機械がテレタイプライタ．テレビジョンと同じく，遠くを表す「テレ」とタイプライタの合成語です．２台の電動タイプライタを１対の電線で接続して，片方のキーボードをたたくともう一方のタイプライタに字が打ち出されるような機能をもった機械です．

デジタル回路や内蔵コンピュータを自由に使えるなら，自由な発想でデータ転送の形式を決められたかもしれませんが，当時はそんなものを使える時代ではありませんでした．ですから，モーターと歯車やクラッチを使って印字する，つまりメカニカルで処理をしなければならないのです．そこで，モールス符号のような可変長の符号体系はあきらめて，固定長の符号体系を採用します．人が操作することを考えれば，つねに一定の速度で文字を送出できないことも多いでしょうし，さらに印字メカニズムが改行するときは時間が必要なので一定の速度で送られても困るわけです．ですから，文字と文字の間の空き時間が変化しても平気な電送フォーマットを決める必要があります．１文字の中の複数ビットの転送時間は一定の方が機械が作りやすいでしょう．

こうして決められた転送方式が，図１に示す調歩同期式とか非同期通信方式と呼ばれる方式です．データを送信する線は普段は１（専門的には１のことをマークと呼びます）になっているとします．データを転送するときには，データの内容とは無関係にまず1bit分の時間だけ０（おなじくスペースと呼びます）にします．このビットをスタートビットと呼んでいます．それから，たとえはデータを7bitだけ転送して，データを転送した後はある一定の時間だけ（例えば2bit分の時間）データ線が１になっていることを保証します．このデータ線が１になっていることを保証している時間を，特別なビットを送っていると考えて，ストップビットと呼んでいます．

このようなデータ転送法でテレタイプライタの内部で何が行われているか見てみましょう．スタートビットを検出した時点で，主動力のモーターとメカニズムの間のクラッチが入ります．モーターは1bit相当の時間で１回転するように調整されています．軸が１回転するごとに，データ線が０か１かで回転軸に接続されたいくつもの歯車が回ったり回らなかったりして，活字の選択機構が動作していきます．そうしてモーターが７回転した時点で，適切な活字が印字準備を終えています．ストップビットを受信している間に，活字をタイプして，クラッチが切れます．また，ストップビットは次のスタートビットをはっきり識別するのにも役に立ちます．

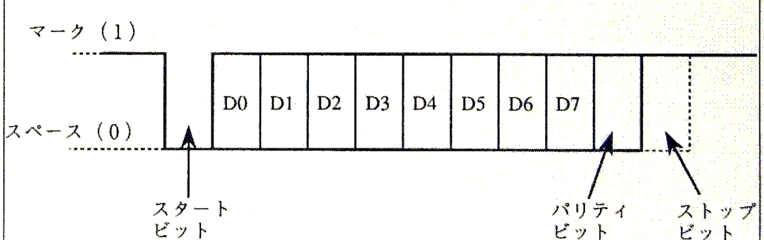
コンピュータとデータ伝送

調歩同期方式のメリット

こうして，メカニズムだけで直列データを解析して印字する機械ができました．ところでコンピュータも同じ方式でデータの送受信を行うインターフェースを採用すると，有利なことがあります．

まず，配線のコストが無視できないほど高価な場合に，トータルコストがパラレル（並列）転送より安くなるということがあげられます．これはデータを送る装置と受信する装置の間が，例えば1000m離れている場合を考えればはっきりわかるでしょう．直列転送なら，２本の電線があれば送信可能ですから，計2000mの電線ですみます．並列転送なら，8bit単位で転送するとしてデータ線が８本，さらに転送のタイミングを取り合うハンドシェーク線と呼ばれる線が２本，計10本の信号線が必要で，さらに転送波形が乱れないようにそれぞれの信号線にグラウン

図1　調歩同期方式



調歩同期式のデータ転送フォーマット．データサイズは５～８bitのときがあるので，D5，D6，D7が省略される場合がある．また，パリティビットも省略される場合がある．ストップビットは，1.5bitや２bit分の時間が使われる場合がある．

ド線が必要ですから，結局合計20本が必要です．トータルの長さは20000m．電線1mあたり10円とすれば，２万円対20万円．直列転送と並列転送では，10倍の差ができたわけです．これが1kmならまだやせ我慢をして並列転送するかもしれませんが，東京と大阪を結んだり，日本とアメリカを結んだりする場合は，とても比較になりません．もちろん，この理由があったらこそ，テレタイプライタが無理をしてでも直列転送をしていたわけです．

さらに，電話回線との親和性があります．いや，元々は電信回線が先にあって，電話機が発明されてから電話回線が広く一般にまで使われるようになったのです．でも，電話回線でもデジタル信号を音声周波数帯の信号に変換すれば送受信できます．つまりモデムという装置を使えばよいのです．いまの高速モデムでは，単位時間に複数ビットを一度に送受信可能にして通信速度を稼いでいますが，以前の技術では単位時間に1bitずつ転送する方式を取っていました．コンピュータの入出力が直列転送を行えるようになっていれば，初期のモデムのおもな回路は，デジタル信号の０と１を特定の周波数の音に変換することと，その逆を行うものを用意するだけです．

最後に，入出力端末として，テレタイプライタがそのまま接続可能になるという点もあります．コンピュータ用のプリンタや端末など，専用装置もあるのですが，テレタイプライタは一般の通信用に量産されていましたから，まだ専用装置より安い場合も多かったのです．ASR-32とかASR-33などのテレタイプの型番を記憶している方もいらっしゃるかと思いますが，それらのテレタイプは本来は通信用だったのです．

専用LSIの登場

CPU対応型に発展

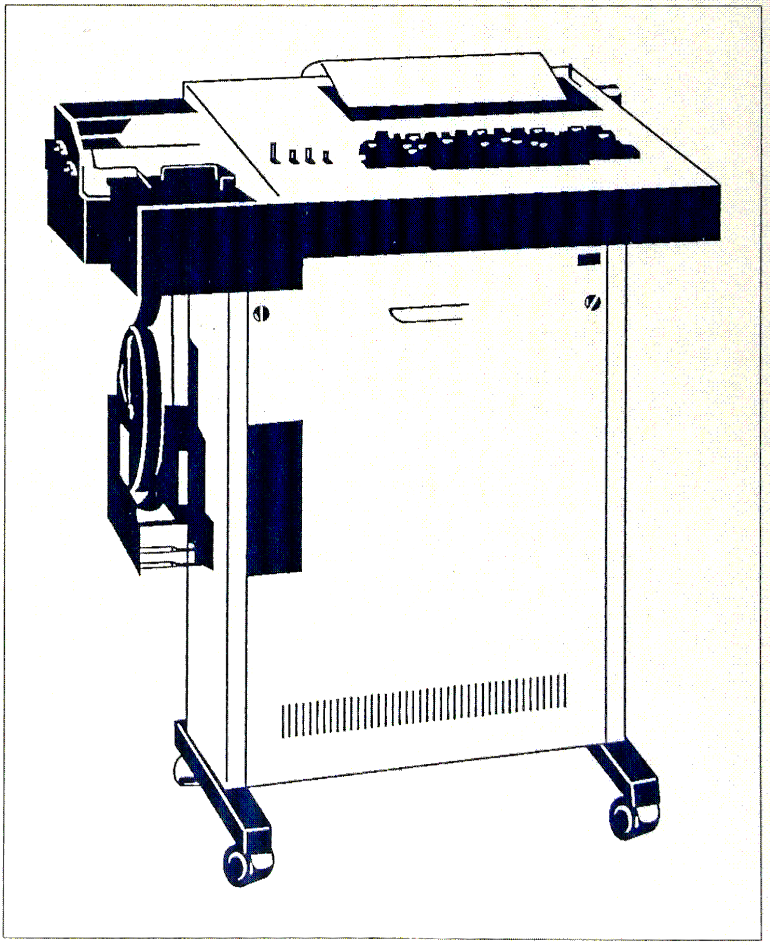
コンピュータとコンピュータ，あるいはコンピュータとほかの機器を，調歩同期方式で結合しようとすれば，そのためインターフェースが必要になります．ほとんどのコンピュータは，内部のバスでは高速の並列転送方式を採用しています．つまり，8bitとか16bit単位でデータをやり取りしています．このデータにスタートビットとストップビット，必要に応じてさらにデータ転送の誤りを検出するためにパリティビットと呼ばれるビットを付加して，1bitずつ送出する必要があります．

この変換を，すべてソフトウェアで行うこともできますが，それでは高速の通信ができないとか，変換作業中はCPUが付きっきりで行うためほかのことがまったくできなくなるなど，多くの欠点があります．

そこで直並列変換を行うLSIが作られました．

最初に生まれたのはUART．Universal Asyncronus Reciever Transmitterの略語です．有名なのはWD1602Aで，P-MOSプロセスで作られていました．電源は5Vと-12Vの２電源です．このセカンドソース品としては，TMS6011とかMB8868などがあります．その後，C-MOSプロセスを使って5V単一電源にしたIM6402

テレタイプライタ

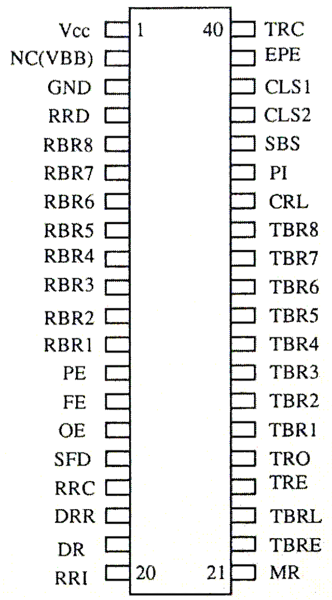


なども使われるようになっています．内部のブロック図とピン配置をそれぞれ図2，図3に示します．

シリアル回線に対して，送信部と受信部が分離しています．受信レジスタの内容は送信レジスタと別々の線になっていますし，各種設定も特定の端子から入力するようになっていました．これは，マイクロコンピュータで操作するためだけに設計されたわけではなく，一般のデジタル回路に組み込むために設計されているからです．このような回路形式なら，パラレル・インターフェースが用意されている入出力機器を，CPUの手助けの必要なく，そのままシリアル・インターフェースとして使えるようにすることができます．

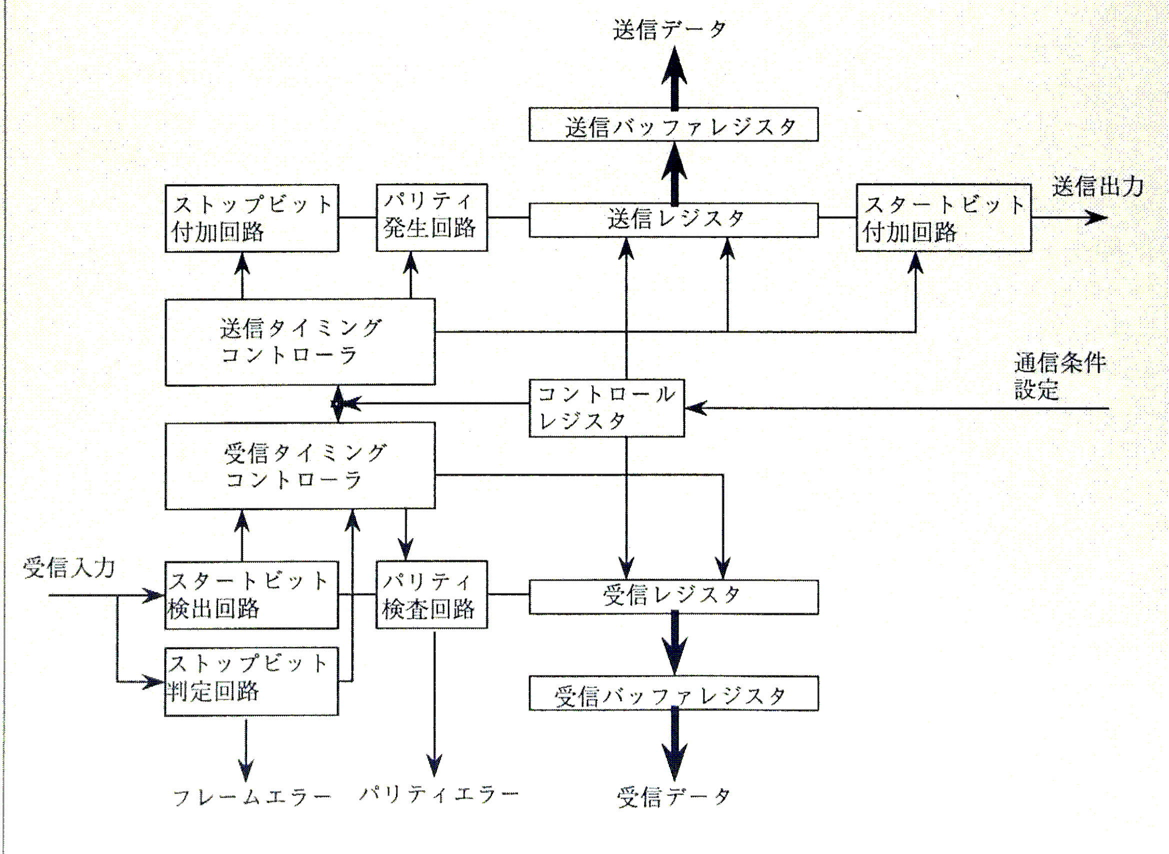
これらUARTは十分に便利だった

図3　UARTのピン配置



RBRnは受信バッファ出力，TBRnは送信バッファ入力で，送受のデータが分離している．

図2　UART(IM6402など)の内部ブロック図



のですが，マイクロコンピュータの入出力インターフェースとして用いるには，不便なところがありました．マイクロコンピュータのバスと直結できるような形式になっていないという点が最大の欠点です．そこでマイクロコンピュータのファミリーLSIとして，主要CPUチップに対応したLSIが使われるようになりました．8080に対しては8251が，MC6800に対してはMC6850が，広く使われていくことになります．

これらは，各種設定も内部レジスタに書き込むことで行われるようになっています．

さらにザイログ社からZ-80 CPUと同時に，専用周辺LSIとしてZ-80 SIOが発表されます．外観は40pinパッケージを使っていますが，2回路分のインターフェースを内蔵しています．さらに調歩同期方式より高度なHDLCなどをサポートする機能も含め，格段に高性能に(また複雑に)なっています．

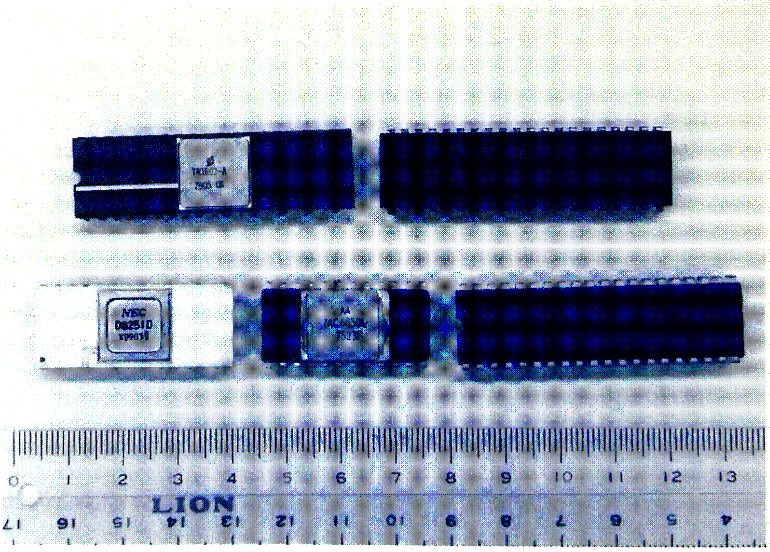
このZ-80 SIOは，Z-80専用として設計されているため8086や68000などほかのCPUに接続するのは困難でしたが，Z-80 SIOを元にそういった一般的なCPUに接続しやすくしたLSIがいくつか作られています．ザイログ社からはZ8530が，また同様の考え方でNECからはμPD7201を始めとしてその改良品が出荷されており，たいへん便利に使われるようになっています．

また，これとは方向が少し別の方を向いていますが，ナショナルセミコンダクタ社のNS8250は，IBM PC系のコンピュータに採用されている高機能シリアル・インターフェースLSIです．

モデムの方は，300bpsなどははるか昔の思い出となり，1200bpsの通信速度のものを使っているところもごくまれにしか見かけませんね．2400bpsが一般的で，回線上のビットレートで9600bpsから14400bpsの速度を持つものが普及しはじめているといったところでしょうか．そろそろ，コンピュータ本体の能力が追い付かない場合もあるでしょう．本体とモデム間のビットレートはそれより大きい速度で行われることを考えると，なによりRS-232Cという規格を決める際に考慮していた速度をはるかに陵駕する速度で利用されているのが驚きでもあります．一応，高速通信用の電気的な規格など，後継と考えられている優れた規格もかなり以前から決められているのですが．

そうはいいながらも，私もいま使っているコンピュータに，RS-232Cのポートがあと3つ4つ付いていて欲しいと思ってます．切り替え器を使っていろいろ接続しているのですが，切り替え器のスイッチは毎回私が使おうとしている周辺機器に対してセットしてあったためしがないのです．

上段左からTR1602-A，IM6402，8251，MC6850，D7201



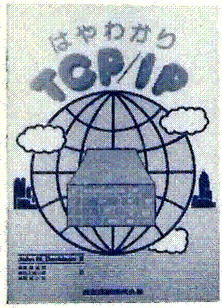
Book Review

■中村正三郎

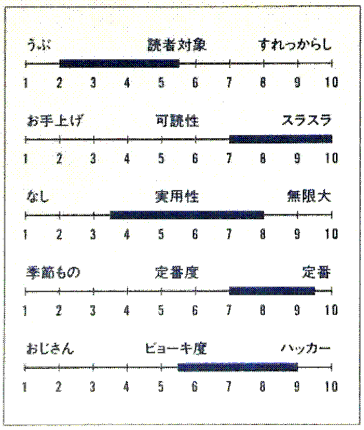
一晩でよくわかるUNIXの標準プロトコルTCP/IP

■はやわかりTCP/IP

John M. Davidson著
後藤滋樹, 村上健一郎, 野島久雄訳
1,900円
共立出版



■評価チャート



「訳者の周辺では，原書が1988年に発行されて以来『一晩でわかるTCP/IP』として重宝がられてきた」(訳者あとがきより)

そのため「一晩でわかるTCP/IP」という書名になりかけたが，誇大広告の恐れありとして「はやわかりTCP/IP」になったという．

大変に誠実な態度で好感が持てる．だが，いやしかし，「一晩でわかるTCP/IP」でもよかった．ほんとに「すぐ効く，よく効く，すぐよくわかる」TCP/IPの解説書である．

評者の場合，一晩ではなく，通勤の行き帰りで読み終えることができた．おさらいできたところ，理解が増したところ，得られたことは大きかった．

原書の題名は「An Introduction to TCP/IP」とそっけないもので，それを「TCP/IP 入門」にしなかったことは，大いに評価したい．なにしろ，なんにでも「入門」とつけてしまう芸のない出版社，著者，訳者が多すぎる．少し「入門」の乱発を控えてもらいたい．

これからTCP/IPを勉強しようという人は，まず本書で土地鑑を養うことをお薦めする．それから規格書や後述の書に挑戦すれば，ずっと理解は容易であろう．

TCP/IPをご存知でない読者のために，本書の話題から離れ，少し説明しておく．

TCP/IPは通信プロトコルの名称，あるいはプロトコル群，さらにはこのプロトコルを用いたさまざまなサービスの総称である．

米国のようにとても広い国土を持つ国であれば，計算機をネットワークしたいと考えるのは当然で，軍事面からもそれは重要であった．1960年後半，米国防総省はパケット交換による長距離ネットARPANETを築いた．今から 20年も昔にナショナルワイドなネットの構築が始まっていたのである．

その過程で検討され，改良され，発展してきたのがTCP/IPプロトコルである．

このプロトコルが世界的に普及し，インターネットの標準プロトコルとしての地位を確立できたのは，UNIX，中でもバークレー版に負うところが大きい．BSD UNIXにはTCP/IPが実装されていたのだ．1980年代を通じ，UNIXの普及とともにTCP/IPも世界中に普及した．

以上のように，TCP/IPの普及はUNIXの普及と強くシンクロしている．裏返せば，UNIX界での標準プロトコルはTCP/IPである．UNIXをOSとして採用しているワークステーションが，容易にインターネットワーキングができるのは，TCP/IPのおかげだ．

ネットワーキングの重要性が高まるにつれ，TCP/IPを知っておくことの重要性も増す．マルチベンダー，異機種，異OSのネットワーク接続の際には，TCP/IPはまず間違いなく登場する．NetWare などほかのプロトコルを採用しているシステムであってさえ，このプロトコルを無視できない．逆にUNIXでも，NetWareやAppleTalkを無視するようなものは，今後生き残れない．マルチベンダーの世界が，それを要求しているからである．

本書で土地鑑を養ったあとに進むべき本としては，規格書を別にすれば，次

の本が教科書,解説書として定番である.

D. E. Comer著『Internetworking with TCP/IP Vol. I : Principles, Protocols, and Architecture, 2nd Ed.』(Prentice-Hall). 邦訳は，bit別冊として村井純・楠本博之訳『第2版 TCP/IP によるネットワーク構築 Vol. I』(共立出版) が出ている. 原書第1版の邦訳もbit別冊として出ているが，第2版だけで十分のはずだ.

Comer先生は，教育を目的としたUNIXライクなOS，XINUで有名な方だ (オリジナルはLSI-11版だがPC版もある). XINUの本全2巻のうち第2巻は，XINUで試みられたインターネットワーキングの本である (邦訳は啓学出版から出ている). 本書はそこから発展した本といえ，書名の通りTCP/IPの考え方，プロトコルの詳細な解説を行っている.

実装面の詳説は，第2巻の『Internetworking with TCP/IP VOl. II: Design, Implementation, and Internals』(Prentice-Hall) にある. これも『bit』別冊として出版の予定ときく.

注目を集めたGNU Emacs関連図書

古田島義和（神田・書泉グランデ）

いま最も注目を集めているのが，GNU Emacsでしょう. 今月の洋書のベストランキングを見ても，第4位にO'Reilly社の「Learning GNU Emacs」(D. Cameron著) の実習書がよく売れています. 和書のベストランキングでは，啓学出版のユーティリティライブラリシリーズ「初めて使うGNU Emacs」(矢吹道郎監修，宮城史郎著) が第1位になりました. 本書はEmacs 18.55とNEmacs 3.3を使いGNU Emacsについてわかりやすく解説しています. そしてこの後に出版されたソフトバンクの「GNUツールボックス」(吉野智興・村上敬一郎共著) も発売から人気上昇中です. 本書はUNIX系のOS上の処理系で開発されたプログラムをMS-DOS上で動作するための移植方法や，日本語を扱えるようにしたNEmacsをX68000上に移植する方法などを中心に書かれています. MS-DOS，X68000ユーザーには必読書といえるでしょう. これだけの人気は，多くのコマンドをもち，拡張性の高さからUNIXの標準エディタとして，また開発者でもあるRichard Stallman氏が多くのユーザーに支持されていることの現れでしょう.

つぎに洋書の新刊をご紹介しましょう. 洋書のベストランキング第3位のPrentice Hall (以下PH) 社「Standard C Library」(P. J. Plauger著) が好調な売れ行きを示す中で，P. J. Plauger氏とJ. Brodie氏の共著でMicrosoft Pressから「ANSI and ISO, Standard C Programmer's Reference」が出版され，こちらもなかなか好調な動きを見せています. 以前紹介したPH社の「The SPARC Architecture Manual」が未入荷のままで，代わりにSpringer Verlag社の「The SPARC Architecture Technical Papers」が入荷. 今月のベスト第10位と健闘しています. SUNといえば，“SOLARIS”のソフト解説書がNew Riders Publishing社から「Inside SOLARIS : SUN OS and OpenWindows」が入荷，続いてAddison Wesley社からも「Writing Applications for the SOLARIS Environmont : A Guide for Windows Programmers」も近日出版される予定です. Computer Science Press社からビッグタイトルが出版されます. Alfred V. Aho氏，Jeffrey D. Ullman氏の両巨頭の「Foundations of Computer Science」で，基本図書としてロングセラーとなることまちがいなしの1冊です. このほかに，「Fundamentals of Data Structures in C」(Ellis Horowitz著，ほか2名) など，今後同社では各分野の基本図書が予定されていますので，随時誌上でご紹介していきますので，お楽しみに.

最後に新進気鋭のIDGの新刊をとりあげます.「InfoWorld Windows 3.1 Secrets 2/E」(B. Livingston著),「PC World Paradox 3.5 Power Programming Techniques」(G. Salcedo著),「OOS for Dammies」(D. Gookin著),「Official XTree MS-DOS & Hard Disk Companion 2/E」(B. Wood著),「Amiga World Official Amiga DOS2 Companion」(B. Ryan著),「Amiga World Official Amiga Vision Handbook」(L. Wallace著) などで，これからの出版が楽しみです.

■和書&洋書ベスト10

（書泉グランデ調べ：1992年2月1日〜2月29日）

	和書	出版社		洋書	出版社
1	初めて使うGNU Emacs	啓学出版	1	Programming Perl	O'Reily
2	オブジェクト指向狂詩曲	技術評論社	2	Simple Book	Prentice Hall
3	Cプログラマのためのアルゴリズムとデータ構造	ソフトバンク	3	Standard C Library	Prentice Hall
4	パソコン界遊学見聞録	技術評論社	4	Learning GNU Emacs	O'Reilly
5	スーパーエンジニアへの道	共立出版	5	UNIX Network Programming	Prentice Hall
6	UNIXオブザーバー1992.3 No.4	スペック	6	Object-Oriented Databases	Prentice Hall
7	カオス―新しい科学をつくる	新潮社	7	Object-Oriented Modeling and Design	Prentice Hall
8	電子立国 日本の自叙伝（下）	日本放送出版協会	8	Object-Oriented Systems Analysis & Specification	Prentice Hall
9	こんなもんいかがっすか	アスキー	9	Eiffel : The Language	Prentice Hall
10	パソコンLANとは何か	講談社	10	SPARC Technical Papers	Springer Verlag

★ ★ ★佐々木哲の★ ★ ★

U.S. プロダクツウォッチ

★偽シェアウェアに関するお詫び

2か月前のこのコラムで，有名なゲーム「SimCityのDOS版がシェアウェアに」としてご紹介いたしましたが，これは全くの誤りでした．お詫びするとともにこの項目を取り消させていただきます．

真相は明らかではないのですが，何者かが市販のSimCityの旧版1.0を1つにまとめ，あるところの古い説明書を付けて，いかにも本当にシェアウェアになったがごとく偽装してからBBSにアップロードしたようです．販売元とされたところは一切その事実を知らないとのことです．

私が見つけたBBSは，それが直接アップロードされたところではなく，さらによそから転送されてきていたようです．アップロードされてからしばらくの間あったので，問題なしと判断したのですがみごとにやられました．USのBBSはかなり巨大化，商業化し，おたがいのメッセージベースを国中あるいはヨーロッパ，オーストラリアなど世界中と交換しあっているだけではなく，Internetなどとも繋がるところも出てきています．ファイルにしても1日に数MBもアップロードされ，人気のあるプログラム類はどこかに登録されると一晩のうちに全米をかけめぐります．

この“善意のシステム”を悪用されてしまいました．読者の皆様また実際にそのプログラムをダウンロードされた方々には申し訳ないことをいたしました．深くお詫びいたします．

★WordStar，Delrinaと合併へ

栄光の英文ワードプロセッサ，WordStarを販売するWordStar InternationalはカナダのDelrinaと合併するようです．WordStar InternationalはCP/Mのころから英文ワードプロセッサとしては確固たる地位をきずいてきましたが，ほかのワードプロセッサ群に押されシェアを失いつつあります．伸びてきたWindowsマーケットにはNBI IncよりLegacyというWindows用ワードプロセッサをそっくり買うことで対応しましたが，うまくいっていません．また，DOS版のものも今ではWordStar for DOS 7.0をリリースするまでになっていますが，なかなか失ったシェアを取り戻すのはたいへんそうです．

一方，フォーム（帳票）作成，管理ソフトPerFormで知られるDelrinaですが，最近Windows用のFAXソフトWinFAXでもその優れた技術を見せています．ところが肝心のフォーム・ソフトのマーケットが予想通りには伸びず苦戦しているのは以前この欄でもお知らせしたとおりです．

技術力，資産をふやし，マーケットシェアをふやし，再建をにらむWordStar Internationalとの合併がうまくいくか見ものです．

★ラップトップマシンも高解像度時代へ

さて，このコラムでも，となりの西川さんのコラムでもさんざんお知らせしてきたVGAおよびSuper VGAの世界．アクセラレータ付きでWindowsなどGUI環境に特化したチップの出現で，来るところまで来た感もあります．4月6日に発売されたMS-Windows 3.1では，今まで本来の性能を発揮できずにいたTIの340x0系など，インテリジェントなグラフィックボードも，その真価を生かせるようになります．グラフィックドライバのインターフェースの改善によるものです．

高解像度化も進み，1024×768は当たり前の雰囲気になってきています．対応するディスプレイもゆるやかながら価格が下がりつつあります．

唯一この流れから取り残されていたのが，LCDなどパネルのディスプレイを使うラップトップ・ノートブックコンピュータです．640×480の壁に押えられていました．

USのNEC Technologies Incはラップトップですが，800×600の解像度を持つ機種，ProSpeed 486SX*1を発表しました．10.4インチのLCDディスプレイに800×

ワンラインNEWS
松下電器産業は64ビットカラーラップトップ型WSを4月末に発売．これは米ソルボーン社のWSで，ラップトップ型では初めてカラーTFT型LCDを搭載．

600の解像度，16階調で表示します．20MHzの486SXと32ビットのEISAバスを１つ持つことにより，デスクトップ並のパワーを備えることになります．

Windows 3.0とMS-DOS 5.0が，120MBあるいは240MBのHDに元からインストールされてきます．AC電源は当然110から240Vまで使用できます．内蔵のMNP 5 2400bpsモデム，あるいは2400/9600bpsのFAX modemはオプションです．

C&TがLCDを1024×768で駆動できるチップを開発したように，これからもラップトップ・ノートブックの高解像度化が進みそうです．

＊1 NEC Technologies Inc. Tel: (508)264-8000
ProSpeed 486SX
メモリ2MB，SIMMにより20MBまで実装可
AC電源のみ
重さ16.71b
120MB HD $6,999
240MB HD $7,499

★もっと軽いノートブックを!

現在，6lb（約３kg）前後が主流のノートブックコンピュータですが，まだまだ重すぎます．それにバッテリ駆動時間も短すぎる．２時間もろくに動かない機械が堂々と販売されているのが実状です．

この春から夏にかけて，より軽い機種が続々と登場する気配です．サブ・ノートブックというべき機種たちです．重さは１kg台から２kgくらいまで．これらの機種は，性能的には現在主流の386SXのVGAノートブックのレベルを保ちつつ，より軽く，使用時間を長くしたものです．

すでにDellは，25MHzの80386SLを使った4lb（約２kg）の機種のテストをはじめたようです．NEC，Toshiba，Epsonなども同じような機種を準備しています．この重量でHDは内蔵，FDはなしとか．さらにAT&Tは美しいデザインで知られるSafariの新機種として，重さ3lb（約1.5kg）で80MBのHDとFD付きの機種を計画中といいます．

本来，386SL CPUの売り文句であった節電技術も今のところ十分生かした機種が見あたりませんが，そろそろバッテリ駆動時間８時間というような，ノートブックが市場に現れはじめています．

★VGA，VGA，VGA

さて，たびたび本欄でもお伝えしてきた高解像度，高速グラフィックカード．そろそろネタが尽きてきました．後は速くなるだけの勝負になってきています．

ひとつ変わったものをご紹介しておきましょう．

640×480という標準VGAの解像度で24ビットのフルカラーを表示できる安価なVGAカードがCardinal Technologies[*2]から発売になりました．新しい24ビットRAMDACを使用することで，WindowsやAutoCAD上で1670万色が同時に表示可能です．VGAチップには，毎度おなじみTseng LabsのET4000を使用しています．高速なドライバを用意したといっていますが，アクセラレータものっていないSuper VGAチップですから，この24ビットフルカラーではそれほど速くはないでしょう．しかし，このボード，定価が399ドルということです．この普通のSuper VGAボード並の値段と普通のVGAディスプレイで24ビットフルカラーが使えることは大きなメリットでしょう．

＊2 VideoColor24 $399
Cardinal Technologies, Inc.
1827 Freedom Rd.,
Lancaster, PA 17601
Tel: (717)293-3000

★ISDNアメリカ流はFBIがらみ

日本では官製の強制普及がはかられているISDNですが，アメリカでは各地方により電話会社が分かれていたり，すでに高速なデジタル専用回線が普及していることもあって，なかなか実用になりません．

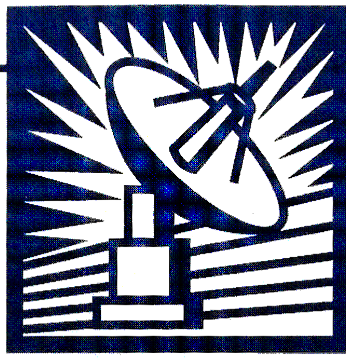
その中で変なニュースをひとつ．音声もデジタルで送られるISDNですが，これはある政府機関，FBIにとっては脅威のようです．すなわち電話回線の盗聴ができなくなる．

そのため，FBIは議会に新たにISDN機器およびソフトウェアに，必ず盗聴を可能にするような機能を盛り込むことを要求する新たな法案を議会に提出しました．

内容は，このためにかかる費用は回線使用料をあげることによりカバーし（つまり回線ユーザーが払う），FCCにそのための回線料金の値上げを認可させるものです．

とんでもないですね……．

Illustration : NOBUKI FUKUHARA



西川和久の

希望的観測

COMPAQがやってきた!!
よくいわれる互換性の問題とは!?

1 COMPAQのプライベートフェア 日本上陸のお披露目

3月3・4日，日本ではじめてのCOMPAQプライベートフェアが行われた．これで海外で有名なPC互換機メーカーは，形式はどうであれ一応日本に参入したことになる．残っているのは，最近個人輸入でよく名前が出てくるGateway程度であろうか!?　日本価格やCOMPAQ-DOS 5.0J/Vなども同時に発表され，準備万端といったところである．気になる値段のほうも，COMPAQ LTE lite/25 model60（A4判ノートタイプ，386SL/25MHz，RAM 4MB，HDD 60MB，日本語キーボード，2.7kg，最大4.5h）で498,000円と，同クラスのPS/55 noteやDynaBook/Vと比べても内容で考えれば割安だ．もちろんサーバモデルである，SystemProはかなりのお値段になってしまうが……（^^;

展示しているマシンをいろいろ触っていておもしろいものをDOS関連で見つけた．基本的にはDOS 5.0/Vなのだが，2つほど拡張しているようだ．ひとつはVE386.EXEである．これはDOS/Vの日本語モードがセグメントB800hにVRAMを持たないため，日本語環境下で海外のアプリケーションを立ち上げた時に何も表示されない不都合(?)を解決するツールである．仮想86を使ってB800hにVRAMを貼り付け，DOS/V自体の仮想VRAMのアドレスもそこへ移動する．実際，DOS5付属のEDIT.COMを使って見たが，半角罫線が化ける以外，何の支障もなく動いている．半角罫線が化ける点も[NumLock]キーを2秒以上連続して押していると，自動的にUS-PC半角キャラクタセットに切り替わる．まるでオンラインソフトのTR，WAVE386，chgfntを組み合わしたかのようであった……VE386.EXEはデバイスドライバなのでCONFIG.SYSに書き込まなければ，普通のDOS/Vになる．

もうひとつはDOSのディレクトリにADDDRV，DEL-DRVを発見した．有名なコマンドなので詳細は解説しないが，これを使うと「かな漢字」などキャラクタデバイスがシステム再起動なしに切り替えることができるのだ．DOS/Vにはないコマンドである．残念ながら短時間しか触っていないので，ひょっとしたらまだ拡張している部分があるかもしれない．早い話，パワーユーザーに受けそうなDOSであった……．

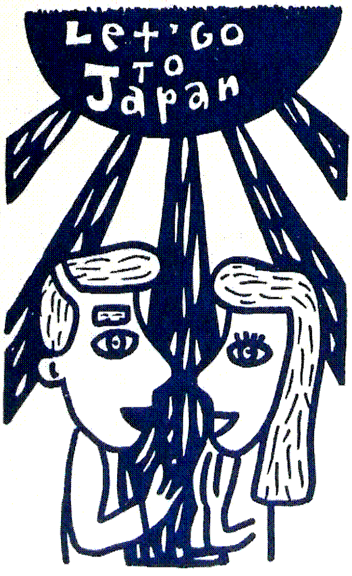
2 互換性はどこが保証する？ 海外での作法に沿って……

先の話からもわかるように，DOSはDOS/Vをベースに拡張されている．そこでユーザーにとっての一番不安なことは「互換性」の問題と「何処が作動保証するか!?」である．当日，かなりのソフトウェアハウスが出展していたため，担当の方何人かにこの点を聞いてみた．どこに聞いても結果は同じ「従来のDOS/Vアプリケーションを動かし問題が出れば，COMPAQ側の責任」だそうだ．また問題があった場合「報告はしても，

あえて動くようにアプリケーションを手直しする気もない」ともいっていた．これは単純明快であるし，私もそれで良いと思う．OADG関係やいろいろなところで，すぐ互換性を指摘する方をよく見かけるが，どうも皆さん「PC-98とEPSON程度の互換性」と意味を履き違えているようだ．

もともと全世界で5000万台もあるPCマシン市場の中でも，かなり有名なメーカーが日本に上陸しているのである．互換性で何かトラブルがあれば，生き残り合戦が激しい海外では，とっくの昔に会社がなくなっている．またソフトウェアベンダーの態度もはっきりしており，パッケージに記述している対応機種は「IBM PC，XT，AT，PS/2 and true compatible」となっているケースが非常に多い．最後の“true”をどこが保証するかは当然ハードウェアメーカーになる．海外では互換性をチェックするだけの会社が何社かあり，「××社の御墨付き」をもらっているケースが多いのだ．

基本的にDOS/Vもこの海外の作法に沿って互換性を保証する．当然リファレンスマシンは絵に書いた餅でなく，PS/55の現行機種がリファレンスになる．たとえば，PS/55 noteで動いて互換機で動かなければ互換機が黒になる．決してソフトウェアが黒にはならないのだ．ただし，PC互換機は後発の日本であるから，OADGというガイドライン引き，コアグループで互換度チェックを行うのだ．



3 しかし問題点が……キーボードが一番のネック

しかし「これで完璧か!?」といえばそうでもない．キーボードとマウスポートの問題が残っている．キーボードはごていねいにIBMのキーボードはもちろんのこと，AX/J-3100/US101キーまでサポート対象にしてしまった．アプリケーション側でのキーボードの見方として，INT 16h，INT 09h，INT 15hがあるのだが，INT 16h以外でキーボードを見てしまった場合，物理的なマッピングになり，互換性が損なわれる．当然，OADGのテクニカルリファレンスにはINT 16h以外を使わないよう指示しているが，現状ですでに使っているアプリケーションや，INT 16hでは拾えないキーコンビネーションがあるため，なかなか難しい問題となる．結果，INT 16h以外を使う場合で全キーボード対応とするには，キーボードドライバを複数持つ必要があるのだ．これは非常にソフトウェア側に負担がかかる．

マウスは通常，PS/2マウスかシリアルマウスがよく使われる．これが曲者で，MOUSE.COMを常駐しINT 33hレベルで制御するとどちらも等価となるが，INT 15hで見るとPS/2マウスしかサポートできない．運が悪いことに最もユーザーが多い一太郎(Dash)，花子がこうなっているのだ．そのほかのアプリケーションでは私の知る限りこのファンクションは使用していない．しかしこの問題は先のキーボードほど深刻ではなく，解決策が2つもある．ひとつはINT 33h対応もしくは，シリアルマウス対応の一太郎を作ってもらう．J-3100版などですでに実績があるから技術的には簡単であろう．もうひとつはINT 33hの割り込みをフックし，INT 15hへ変換するTSRを作れば良い．解決策があるから，気が楽である．(^^;

以上の分析から，一番のネックはキーボードとなる．これ以外はとくに問題ないであろう．考えてみれば，世界的に見てもキーボードは各国々でいろいろな仕様があり，ここ数年，海外でもできるだけINT 16hでキーボードを見るようソフトウェアハウスは努力している．すなわち，流儀に従うことで自社の負担を減らしているのだ．「作法を守ろう!!」これが全国共通の合言葉なのだ．

Computer Laws

コンピュータの法律

コンピュータプログラムの登録制度について

熊谷信太郎（弁護士）

■著作権取得の登録

コンピュータプログラムの中には，ほとんど財産的価値がないと思われる趣味用のものもあれば，国の防衛や企業の財務システムなどに関する，莫大な財産的価値のあるものもあります．価値のあると思われるコンピュータプログラムには，大金を投資してでも他社に先んじようとする企業は少なくありません．

しかし，コンピュータプログラムの製作者（すなわち著作権者）とのプログラム著作権の譲渡契約に成功し，多額の金銭を支払って著作権を買い取ってプログラムを使用できるようになったとしても，それだけで安心というわけにはいきません．コンピュータプログラムの著作権取得を登録しておく必要があるのです．

たとえば，ソフトハウスのA社が開発したコンピュータプログラムの内容が画期的で，他社のものより優れているため，B社からA社に対し1億円で買いたいという申し出があったとします．A社はB社の申し出を承諾して，1億円でコンピュータプログラムの著作権を売る契約をしたのですが，その後に，同じ業界のC社から1億5000万円でそのコンピュータプログラムを買いたいという申し出があったので，A社は先に契約したB社を裏切ってC社にコンピュータプログラムを二重に売ってしまったという場合，B社とC社のどちらがコンピュータプログラムの著作権を取得するのでしょうか．

■契約順か，登録順か

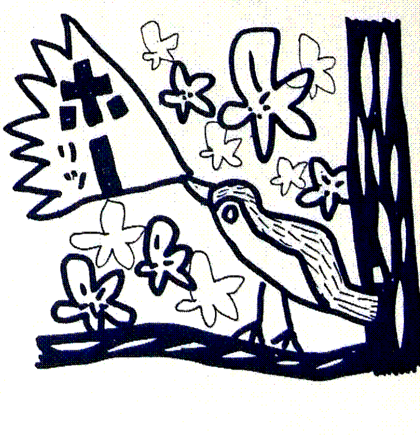
読者の皆さんの中には，当然先に契約したB社が優先するに決まっていると思った人も多いのではないでしょうか．

しかし法律的には，どちらが先に著作権の譲渡契約をしたかは，B社とC社の優劣には関係ないのです．譲渡契約の先後に関わりなく，先に文化庁にある著作権の登録原簿にコンピュータプログラムを登録したほうの会社（または個人）が著作権を取得することになっています．

したがって，いまの例の場合，もしC社が先に文化庁で著作権の登録をすませてしまうと，いくらB社が先に契約をしていても，A社に代金を支払っていたとしても，コンピュータプログラムの著作権はC社が取得するのです．B社としてはいくら悔しがっても後の祭りですので，大事なコンピュータプログラムの取引の時には登録を忘れてはいけません．

もっともこの場合，A社はB社にいったんコンピュータプログラムを譲渡しておきながら，場合によっては代金をもらっておきながら，C社に同じものを売ってしまったのですから，道義的な責任が生ずるのはもちろんのこと，法律的にもB社に対する債務不履行になり，B社から損害賠償を請求されることになります．

B社の損害として考えられるのは，A社に支払った代金のほか，そのコンピュータプログラムを他社に転売する予定だった場合の転売利益．たとえば，B社がD社にそれを2億円で売ることになっていた場合には，2億円から取得原価の1億円を差し引いた1億円が損害ということになり，これをA社に対して賠償請求することができます．しかし，A社にそれだけの賠償資力がない場合もあるので，B社がかならずA社からお金をとれるというわけではありません．B社としては，そのような事態に追い込まれないようにきちん



Illstration : Nobuki Fukuhara

と登録しておくべきなのです.

■プログラムも担保になる

また，法律的にはコンピュータプログラムの著作権も土地や自動車などと同じく，譲渡できる財産権ですので，それを担保にして質権を設定し，お金を借りることができます.

コンピュータのソフトハウスなどで，土地などの担保になる資産がないのでお金が必要な時に融資が受けられないという場合がありますが，もし，ソフトハウスが資産性の高いコンピュータプログラム，すなわち皆が欲しがるようなコンピュータプログラムをもっている場合には，それを担保にして金融機関から融資を受ければいいのです．ただし，金融機関はコンピュータプログラムの価値についてよく理解していないのが現状ですから，その点はよく説明して，担保価値があるということをわかってもらわなければなりません.

この場合にも，著作権法の規定にもとづき，質権設定の登録をしておかないと，そのコンピュータプログラムについて質権を有していることにつき第3者に対抗できないので，注意が必要です.

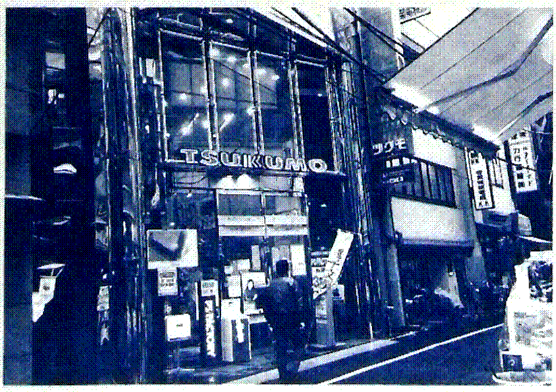
アキバ短信

ツクモパソコン本店

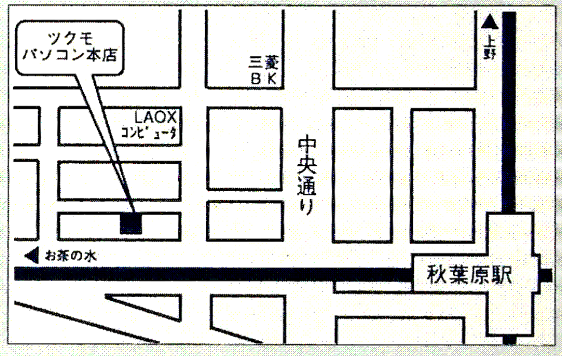
本体・ソフトから書籍まで揃う機種別のフロア分け

秋葉原電気街の特色のひとつは，多店舗で展開しているショップが多いことだ．ヤマギワ，石丸電気，LAOX，ロケット，サトームセン，ナカウラetc.．これらの店で待ち合わせようと思ったら「石丸電気のマイコンセンター」だとか，「サトームセンの中央通り店」だとか，「ヤマギワのリビナ2」のように，フルネームで確認しておく必要がある(もっとも待ち合わせには，もっとわかりやすい場所を選んだほうがいいことも確かだ).

九十九電機もまた秋葉原に多数の店舗をもつ古顔の電機店だが，「九十九」という少々古めかしい名前を改め，最近では「ツクモ」に統一している．老舗ツクモがパソコンの専門店として2年前にオープンしたのが，「ツクモパソコン本店」だ．この店の特徴は，なんといっても地下1階から地上3階までの4フロアが完全に機種別に分けられていることだ．1階がNEC PC9800シリーズとEPSON互換機，2階がシャープX68000と富士通FM TOWNS，3階が



営業時間　10:00〜19:00
TEL　03-3253-5599
定休日　木曜日



IBM PCとDOS/V関係，地下1階はMacintoshとNeXTで，パソコン本体，ソフトウェア，周辺機器から書籍，雑誌に到るまで，その機種に関連するものがすべてひとつの売り場にまとめて展示されているのだ．なかでも，IBM製品には力を入れており，3階にはIBM製品のシステム的な相談にも応じてくれるPS FACTORYを開設している.

もうひとつの特徴は，TSシリーズというオリジナルの増設フロッピーディスクドライブを販売していることだ．98，EPSON用だけでなく，X68000シリーズ用，J-3100用，IBM PS/55用など各機種用の製品がある.

ツクモパソコン本店の入り口は，LAOXコンピュータ館の裏側，秋月電子のある通りと，ラジオデパートのある線路沿いの通りの間にある通りに面していて，あまりわかりやすいとはいい難い場所なのだが，立地条件のマズさをカバーするにあまりある派手なネオンの看板を掲げている．また，LAOXのニュー楽器館と同じ並びにあるツクモ7号店とは1階の奥でつながっているので，ここから入ることもできる．ちなみに7号店のほうは2階がMIDI関係の製品を集めた売り場になっているので，こちらにも注目したい.

●プログラマの仕事場 | 14

PW Programmer's Working Space

日本ワードパーフェクト

米国生まれで米国育ちの日本語ワープロソフトがやってくる

欧米で約900万のユーザーを有するWordPerfectが，完全な日本語化を実現し，日本語ワードプロセッサとして市場に登場した．アメリカ製の日本語ワープロは日本の市場でどこまで受け入れられていくのだろうか．Word Perfectの日本支社を訪ねて，マーケティングリーダー徳田英樹氏にお話をうかがった．

ミニコンのタイプライタ代わりから市場占有率ナンバー1へ

現在では市場占有率50パーセント超をほこるWordPerfectだが，その歴史は意外に古い．誕生は1979年．米国ユタ州のオレムという小さな街の市役所の文書処理システムの一環として，データゼネラル社のミニコン用に作られたのがはじまりである．プログラムを書いたのは，この街にあるB. Y. U.という大学の教授とその学生２人であった．1981年にIBM PCが発売されると，このソフトをPCに移植する．とはいえ，当時のワープロは作成した文書をファイルに保存することができるという点をのぞけばタイプライタと同程度の機能しかなかった．

ところが，これから数年の後，やはりB. Y. U.の学生の１人によってスペルチェッカーという画期的な機能がつけ加えられて，徐々に売り上げを伸ばしていく．このスペルチェックプログラムを開発した学生が現在も日本語版の開発に携わっている同社のスチュアート・スミス氏だ．

WordStar，Multimate，Display Wrightという強力なライバルの陰に隠れて，最初は目だたない存在だったWordPerfectだったが，地道にバージョンアップを重ね，Ver4. 2でついに前年比２倍の売り上げを達成する．WordPerfect社自らが大規模なコンファレンスを開き，ユーザーを招いてその要望を聞き，WordPerfectがユーザーのニーズに見合う機能を備えていることを強力にアピールしたのが効を奏したのだという．そこは，さすがにプラグマティズムのお国柄だ．

1988年に発売したVer. 5. 0でついにWordStarを抜いてワープロ市場で第１位となる．その後89年末に発表したVer. 5. 1が最新バージョンである．DOS版のほかにも，Windows版，Mac版，各種UNIXワークステーション用，IBMのメインフレームである370用，AMIGA，ATARI，AppleII用など各種コンピュータ用に移植されている．移植のベースとなるバージョンは機種ごとに違いがあるものの，これだけ多くのマシンに移植されたソフトも少ない．

ワンラインNEWS
日商エレクトロニクスは処理能力を向上させたUNIX対応ファイルサーバ「NS5000」，「同3000」を発売した．同機は米オースペックス社の製品．

アメリカ人がアメリカで作った日本語ワープロ

そのWordPerfectが今度は日本語ワープロWordPerfect5.1Jとして，日本で発売される運びとなった．ところが，驚くべきことに日本語ワープロ化にあたっては，ほとんどの作業がアメリカで行われたのだという．

「開発はほとんどアメリカのユタ州でやっています．ご存知かと思いますが，ユタ州というところはモルモン教徒の多いところで，WordPerfect本社のあるオレムも街の人口の90パーセント以上がモルモン教徒です．B.Y.U.という大学もモルモン教の教会がバックになっていて，タレントのケント・ギルバートやケント・デリカットもここの出身です．NetWareで知られるノベル社もあります.」

「日本語版を開発しているチームは全部で20数名ですが，全員日本語が話せるんですね．全員がモルモンの宣教で日本に来たことのある人ばかりなのです．ですから，マニュアルも全部読むことができますし，ほかの者もプログラマだけあって言語には長けているようですね．開発チームでは皆，漢字や日本のものには愛着があるらしくて，銭湯ののれんだとか，日本のカレンダーなどが飾られていて，その一帯が『日本村』といった趣になっています」(徳田氏).

2バイト文字からFEPの搭載まで日本語ワープロ化の苦労

とはいえ，英文ワープロと日本語ワープロとでは，1バイト文字と2バイト文字の違い，入力の方法や，行末の処理の方法などにもかなりの隔たりがある．移植にあたっては，苦労も多かったようだ.「まずは2バイト文字を通すまでがたいへんでした．WordPerfectでは，『このポジションから文字の大きさを変える』『ここからは文字をボールドにする』といった情報に1バイトを充てていたため，文字そのものが2バイトになるとその情報をどこでもつのかが大きな問題となってしまったようです.」

「日本語ワープロ化するにあたって，もうひとつ大きな問題となったのは，日本語入力をどうするかです．自社で1から開発するのはとうてい無理だろうということで，今回の製品はエルゴソフトさんからソースコードライセンスを受けて，それをWordPerfect用に変更しました．なぜそうしたかといいますと，MS-DOS版のWordPerfectはすべて［SHIFT］＋ファンクションキー，［CTRL］＋ファンクションキー，［GRPH］＋ファンクションキーで操作するようになっているのです．ですから，VJEやATOKのような既存のFEPを組み込むと，VJEなら［CTRL］＋ファンクション，ATOKでは［SHIFT］＋ファンクションがFEPとぶつかって，FEPに制御を取られてしまうのです．それに98と，東芝，IBM，DOS/Vの全部に対応する必要がありましたから，他社のFEPをOEMで使うとなるとそれだけで膨大な金額になってしまうという問題もありました．そこで，自社で変更できるよう，ソースコードライセンスを提供してくれるところが必要だったのです．ただ，辞書の内容にまで手を入れることができませんでしたので，変換水準はエルゴソフトさんのEGBridgeと同レベルです」(徳田氏).

About Company

日本ワードパーフェクト．正式にはWordPerfect International Inc.（米国ユタ州）の日本支社．支社長ジェフリー・ルイス．社員数は'92年3月現在27名．



米国本社

Location

営業と業務部門は，JR山手線の恵比寿駅，東急東横線の代官山駅からほぼ等しい距離に位置する．また完成してまもないオフィスビルの4階に，技術・サポート部門は，代官山駅に近い閑静な一角にあるビルの地下1階にある．

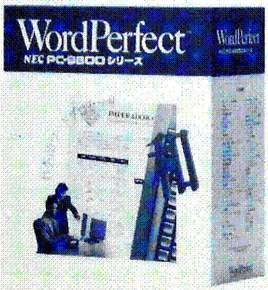
Products

「WordPerfect 5.1J」(58,000円，追加ライセンスパッケージ・マニュアル付き44,000円，同マニュアルなし34,000円，20ユーザーパッケージ580,000円)

全世界での累計出荷本数830万本を誇るワードプロセッサWordPerfectの日本語版．英文編集専用ではなく，完全に日本語の入力・編集にも対応した日本語ワードプロセッサとしての機能を持つ．さらに，TeXライクな数式エディタ，表計算機能を内蔵した自動作表機能，英文スペルチェッカー，Lotus1-2-3やExcelのワークシート／グラフの直接読み込み，PIC，TIFF，MSP，EPS，DXF/HPGLなど各種形式のグラフィックスファイルの読み込み，LipsIIIやPostScriptなど高機能ページプリンタのネイティブな言語への対応など，本格的な機能を備えている．

日本語FEPとして，ATOK6/7，VJE-β，EGBridge，NEC-AI，IBM MKKなど各社の製品に対応，またWordPerfectとのキー操作の親和性を考慮した独自の日本語FEP，WPFEP Ver.1.0も付属する．

NEC PC-9800シリーズ用，東芝J-3100シリーズ用，IBM PS/55シリーズ用，DOS/V用の4種類のパッケージがある．



「WordPerfect 2.1 for Macintosh」(58,000円)
DOSの5.0版を元に開発されたMacintosh用日本語ワードプロセッサ．日本語ワードプロセッサ機能に加えて，ドローイングツールを内蔵．Macintoshのマルチメディアを実現する最新技術である「Quick Time」にも対応している．対応機種はMacintosh Plus，Classic，SE，IIシリーズ．漢字Talk6.0.4以上で動作する．

「WordPerfect5.1」(98,000円，LAN/MultiPaks 5ユーザー285,000円，10ユーザー980,000円)
IBM PCとその互換機上で市場占有率第1位を誇る英文ワードプロセッサ．NEC PC-9800シリーズ用，東芝J-3100用(日本語DOS版)，IBM PS/55用（日本語DOS版)，IBM PC用（英語DOS版）がある．

こうして完成したのが，WordPerfect5.1J専用のかな漢字変換システムWPFEP Ver.1.0であるが，他社のFEPも組み込み可能となっているという．他社のFEPを組み込んで使用する場合には，FEPとWordPerfectのキー操作がぶつかった場合にどちらを優先するかを選択する機能を付加して，ユーザーの便宜をはかっている．

英文ワープロから日本語ワープロへ 日本法人のあゆみ

そもそも，英文ワープロを日本語ワープロとして移植するという大胆な試みはいつごろから生まれてきたのだろうか．

WordPerfectの日本法人ができた1988年にはまだIBM PCが日本国内ではほとんど使われておらず，英文ワープロとしてのWordPerfectは，せいぜい月に100本程度しか売れなかったという．当時の販売先は外資系企業などごく一部のユーザーに限られていたが，米国でVer.5.0が大ヒットしたことから，日本でも少しずつ売れはじめたのだそうだ．

89年には英文ワープロとしてNEC PC-9800シリーズ用に移植，そのころから完全な日本語化の計画がはじまる．日本語ワープロでは必須ともいえる罫線機能が米国Ver.5.1でサポートされるようになった時点で，Ver.5.1をもとにした日本語ワープロの開発が決定したのだという．最終的に'91年末に完成するまで，丸2年をかけて移植したことになる．

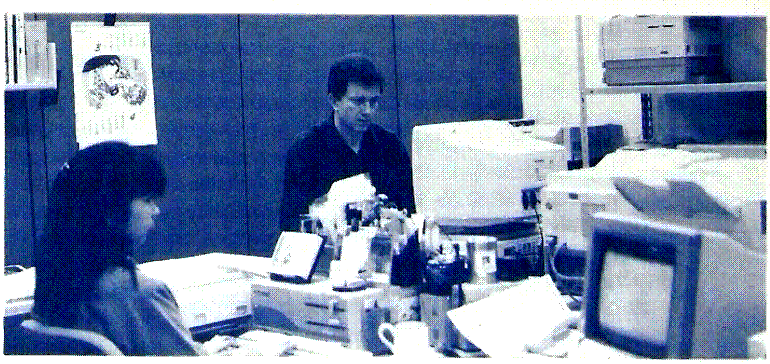
ユーザーサポートに尽力 いずれは12時間体制へ

現在WordPerfect日本支社は，社員数27名，そのうち約半数が技術スタッフである．技術スタッフの仕事は，サポート，翻訳，製品テスト，プリンタドライバの開発などだ．

なかでも同社が力を入れているのがサポートとプリンタ対応である．WordPerfectの米国本社では，1000人ものユーザーサポートスタッフを抱え，24時間無料でサポートを行っているという．「そこまでの対応はできませんが，せめて朝の8時から夜の8時までくらいの時間帯でサポートできるように検討中です」(徳田氏)．日本支社のサポートスタッフは6名だが，これをさらに増強するため現在も募集中とのことだ．

また，プリンタドライバについては，新製品が出たり，何か問題が生じたとサポートから連絡があった場合にすぐに対応できるよう，日本国内で開発を行っている．ワープロにとって，印刷機能は非常に重要だと位置づけているからだ．すべてのプリンタについて，ネイティブモードで対応し，プリンタの機能を最大に利用した印字品質をめざしている．

WordPerfect社ではこの日本語対応版を在米の日本企業をターゲットに米国内でも販売する予定だという．もっともよく使うだけに，競争もはげしい日本語ワープロソフトの世界で，WordPerfectがどれだけ健闘していくのか，今後が楽しみである．



●イベント

■4月22日(水)～4月25日(土)
マイクロコンピュータショウ'92
半導体、電子デバイスから応用機器、ソフトウェアシステムまでを含むマイクロコンピュータに関する総合的な展示会。
会場：東京流通センター大展示場(東京・平和島)
入場料：無料
問い合わせ先：(社)日本電子工業振興協会
TEL 03-3433-4547

■4月22日(水)～4月25日(土)
EDUCATION JAPAN
コンピュータや各種メディアを利用した学校教育、企業内・社会人教育のための機器を展示する「CAI&教育メディア・ショー'92」、「ミュージック・エデュケーション・ショー'92」、「キャリア・マネジメントフェア'92」が行われる。
会場：サンシャインシティ展示ホールA・B(東京・池袋)
入場料：500円
問い合わせ先：EDUCATION JAPAN運営事務局
TEL 03-3243-9082

■5月20日(水)～5月23日(土)
ビジネスショウ'92TOKYO
「オフィスイノベーション――より人間らしく快適に」をテーマに、事務処理・情報処理システム、ソフトウェアを展示。
会場：東京国際貿易センター(東京・晴海)
入場料：無料(ただし5月20日・21日の招待日は招待券が必要)
問い合わせ先：(社)日本経営協会
TEL 03-3403-1331

●セミナー

■5月21日(木)～5月22日(金)
SQLWindowsテクニカルセミナー
Windows上のビジュアルプログラミングを実現するSQLWindowsの基礎とプログラミングを紹介する。
受講料：100,000円
問い合わせ先：三菱商事(株)技術第一部
ネットワーク取引開発タスクフォース
TEL 03-3210-7358

Microsoft University

■5月11日(月)～5月15日(金)
Microsoft Windows Programming Environment
Windowsアプリケーション開発のための入門コース。
受講料：180,000円(5日間)

■5月11日(月)～5月12日(火)
C言語入門
Quick Cを使ったC言語の入門コース。
受講料：60,000円(2日間)

■5月13日(水)～5月15日(金)
C言語応用
MS-C 6.0を使ってC言語の高度な技術を学ぶ実習中心のコース。
受講料：90,000円(3日間)

■5月18日(月)～5月22日(金)
Microsoft Windows Applications Programming
Windowsアプリケーション開発のための上級コース。
受講料：180,000円(5日間)

■5月11日(月)～5月15日(金)
MS OS/2 Applications Programming
MS OS/2アプリケーション開発のための上級コース。
受講料：180,000円(5日間)

■5月25日(月)～5月29日(金)
MS OS/2 Presentation Manager Applications Programming
MS OS/2 PMアプリケーション開発のための上級コース。
受講料：180,000円(5日間)

Microsoft Universityは、OS/2、Windowsアプリケーション開発者のために定期的に開催されているセミナーです。
問い合わせ先：富士ソフトウエア(株)
品川教育センター　TEL 03-5461-8296

NEWS & NEW PRODUCTS

ボーランド，98用BC++ & AFX3.0，TC++ for Win3.0を発売

ボーランド（株）（TEL 03-5350-9365）は，PC-9800シリーズ用「BORLAND C++ & APPLICATION FRAMEWORKS 3.0」（以下BC++ & AFX3.0）と「Turbo C++ for Windows 3.0」（以下TC++ for Win3.0）を発売する．「BC++ & AFX3.0」は，TC++ for Win3.0を付属，大規模なアプリケーション作成を可能にするプロテクトモードのコンパイラ，メニューやダイアログ，アイコン，カーソルなどの管理，編集，作成ができるWindows用のリソースエディタResource Workshop，Windows用の画面制御クラスライブラリObject Window，DOS用の画面制御クラスライブラリTurbo Vision，ランタイムライブラリソースコード，Application Frameworksソースコード，Turbo Debugger，Turbo Profiler，Turbo Assemblerが付属している．また，ANSI CとAT&T C++ 2.1をサポートしている．「TC++ for Win3.0」には，Turbo Debugger 3.0 for Windows，Resource Workshop，Object Windowsなどのwindowsアプリケーション開発に必要なツールが含まれている．BC++3.0は，DOSのCUIで動作し，DOSとWindows両方のアプリケーションを開発することができる．TC++ for Win3.0は，Windows上で動作し，Windowsアプリケーションのみを開発することができる．価格はBC++ & AFX3.0が98,000円，TC++ for Win3.0が29,800円．出荷はBC++ & AFX3.0が4月下旬，TC++ for Win3.0が5月下旬の予定．

ライフボート，WATCOM社と提携，WATCOM C/386などを発売

（株）ライフボート（TEL 03-3293-4711）とカナダWATCOM Systems社は，WATCOM Systems社の開発したインテル系32ビットチップ用プロフェッショナル向けコンパイラ「WATCOM C/386」と「WATCOM FORTRAN77/386」における日本語版の総代理店契約を締結，これにともない国内機種への移植，日本語化作業を両社が共同で行うことに同意，北米版同様，日本語版においても，DOSエクステンダをはじめとするインテル系32ビット環境のさまざまなプラットホームに対応する32ビットコンパイラの開発と製品化をはかっていく．「WATCOM C/386」は，80386/486プロセッサがもつ32ビットネイティブモードのオブジェクトコードを出力，コンパイラ自身が32ビットネイティブモードで動作するため，従来よりもはるかに大きなソースファイルを高速にコンパイルすることができる．価格は，32ビットDOSエクステンダ「DOS/4G」に対応した「DOS/4G版WATCOM C/386」が198,000円（DOS/4Gとのバンドルに限る），組み込みシステム専用のROM化対応製品「WATCOM C/386 ROMABLE」が298,000円．「DOS/4G版WATCOM FORTRAN77/386」は，最大4GBまでの大規模なアプリケーション開発が可能．「DOS/4G」に対応している．価格は298,000円．両製品とも対応機種は，PC-9800，PS/55シリーズ，PC/AT互換機など．

ライフボート，逆アセンブラ「SOURCER」の新バージョンを発売

ライフボート（株）（TEL 03-3293-4711）は，MS-DOS用の各種プログラムを直接解析し，アセンブラソースリストを生成する，高機能逆アセンブラソフト「SOURCER」の新バージョン「同Ver.4.5」を発売した．対応機種はPC-9800（MS-DOS 2.11/3.X/5.0），FM R，J-3100（MS-DOS 3.X/5.0）などで，98用SOURCERは，従来のDOSファンクションコールに加え，98のハードウェアに依存したソフトウェア割り込み（BIOSコール），I/Oポートアクセスに関する手続きを検出，解析スピードも約5倍の高速化を実現した．出力ソース形式は，従来のアセンブラ処理系に加え，MS-C 6.0，TURBO ASSEMBLER 2.5/3.0を新たにサポートしている．価格は38,000円．

ワンラインNEWS
アルプス電気は8色のカラー表示が可能なSTN型LCDを開発．6.85インチで320×284ドットの表示が可能．製品化までには16色に対応できるようにする．

NEWS & NEW PRODUCTS

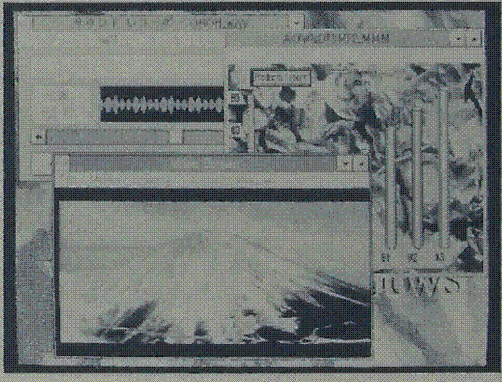
米国マイクロソフト，Microsoft Windows 3.1を発売

米国マイクロソフト社は,「Microsoft Windows 3.1」(以下Windows 3.1) を全世界に向け出荷開始した．Windows 3.1は，昨年5月に発売したWindows 3.0をアップデートしたもので，より速く，容易に使用できるような機能強化点を1000以上含んでいる．同時に6か国語(米国，オランダ，ドイツなど) バージョンを出荷する．なお，日本語版は今年末発売の予定．

富士通，TOWNS用マルチメディアプラットフォーム「Windows MME」を提供

富士通 (株) (FM TOWNSインフォメーションサービス：TEL 03-5690-0032) は，「日本語Microsoft Windows V3.0」(以下Windows 3.0) 上でマルチメディアを利用する国際標準プラットフォームとして，マイクロソフト社の開発した基本ソフトウェア「日本語Microsoft Windows V3.0 with Multimedia Extensions V1.0」(写真，以下Windows MME) を同社のFM TOWNS用にサポート，国内ではじめて販売を開始した．「Windows MME」は，Windows 3.0の機能をベースに，各種音声データの録音・再生，アニメーションの再生，外部AV機器の制御などのマルチメディア拡張機能を加えたもので，標準インターフェース「MCI (Media Control Interface)」を備え，マルチメディアの複合データの統合格納形式「RIFF (Resource Interchange File Format)」をサポートしている．価格は34,800円で，CD-ROMで提供される．

また同時に，40MBハードディスクを内蔵したディスプレイ一体型「FM TOWNSⅡモデルUX40」(価格：388,000円) も発売した．



Windows MME

日本IBM，クライアントサーバシステム強化で新機種発表

日本アイ・ビー・エム (株) (TEL 03-3586-1111) は，クライアントサーバの実現を支援するため，パソコンLAN構築にかかわる計画，導入，運用などの一連のサービスを一貫したトータルソリューションとして提供する「クライアント／サーバービジネス」を開始した．その一環として，他社製パソコンも含めたパソコンLANの接続性テスト機関である「マルチ・ベンダー・コネクティビティ・センター」を同社の飯倉事業所内に開設する．同時に，PS/55ファミリーの最上位モデルで，サーバとして大量データの高速処理が可能な「IBM PS/55モデル5580-Y1」と同ファミリーの中核に位置する「同モデル5551-W0」の2モデルを発表．「モデル5580-Y1」は，80486 (33MHz)，標準で8MBのメモリ (最大32MB) と400MBのハードディスクを搭載，SCSIアダプタを標準装備している．価格は2,400,000円(5580-Y1C)．「モデル5551-W0」は，80486SX (20MHz)，標準で4MB (最大28MB) と100MB (5551-W0A)・200MB (5551-W0B) のハードディスクを搭載している．価格は，5551-W0Aが950,000円，5551-W0Bが1,050,000円．また，昨年10月に発表した，10.4インチの大画面TFTカラー液晶ディスプレイ，80386SX (20MHz)，80MBハードディスクを搭載したノート型パソコン「PS/55noteカラー・モデルN27sx」の一般供給も開始した．OS/2 V2.0 (VGA対応) とIBM DOSバージョンJ5.0/V (DOS/V) に対応している．

ワンラインNEWS
韓国の三星電子はトーメンと共同出資で日本国内に半導体の販売会社を設立する．これまで販売は商社に委託してきたが，これでシェア拡大を目指す．

NEWS & NEW PRODUCTS

日本NCR，日本語手書き入力パソコンを発売

日本エヌ・シー・アール（株）（TEL 03-3582-6111）は，国内初，パーソナルコンピュータの機能をフル装備したペンベースの本格的な日本語手書き入力パソコン「NCR3125ペン・コンピューター」を発売した．従来のキーボードに代わって，電子ペンで直接ディスプレイに入力するノートパッド型のコンピュータで，A4サイズ（248mm×297mm×25mm），重さ1.8kgの小型・軽量化をはかった．80386SL（20MHz），4MBのメモリ，20MBのハードディスクまたは8MBのFEPROM，VGA互換の液晶ディスプレイを搭載，OSにはDOS 5.0を採用，CIC（Communication Intelligence Corporation）の多言語手書き文字認識システムとペン・インターフェース技術を内蔵したPen DOSを組み込んでいる．価格は823,000円．

金亀電子・大同日本，DOS/V対応IBM PC/AT互換の新シリーズを発売

金亀電子（株）と大同日本（株）の両社は共同で，80486搭載モデルなどデスクトップ型２機種とノート型１機種のDOS/V対応，IBM PC/AT互換機の新シリーズを発売した．最上位機種「AEGIS430DH」は，日本語SVGA，S3社製高速VGAカードを使用，CFC社の日本語Windows高解像度ドライバを標準装備している．また同時に金亀電子（株）は，80486（33MHzと50MHzタイプあり）を搭載したNetWare用サーバ専用機「スーパーサーバシリーズ」を発売した．これはモニタ装置付き無停電電源を搭載し，ハードディスクのインターフェース，ドライブを二重化するなど信頼性を重視，ビデオボードには日本語版NetWareをサポートするJEGAボードを用意，上位機種のPOWER-1にはEISAバスを採用している．価格は，7,860,000円（POWER-1），4,960,000円（POWER-2）．

■AEGIS430DH

プロセッサ：80486DX（33MHz）／メモリ：4MB／外部記憶装置：3.5インチFDD×1・5インチFDD×1・200MB HDD×1／価格：488,000円

■AEGIS420SH

プロセッサ：80486SX（20MHz）／メモリ：4MB／外部記憶装置：3.5インチFDD×1・5インチFDD×1・100MB HDD×1／価格：298,000円

■AEGIS320NB

プロセッサ：80386SX（20MHz）／メモリ：4MB／外部記憶装置：3.5インチFDD×1・40MB HDD×1／ディスプレイ：CCFTバックライト付きFTN液晶ディスプレイ／価格：295,000円

アークブレイン，Polywell社のIBM PC/AT互換の新機種を発売

（株）アークブレイン（TEL 03-3375-8968）は，米国Polywell社のIBM PC/AT互換の５種の新機種を発売した．英語版のMS-DOS Ver.5.0がすでにハードディスクにインストールされており，DOS/V，日本語Windows，OS/2などを動作させることができる．80486（50MHz）を搭載したハイエンド機「486-50VM」は，8MBのメモリ，グラフィックカードにはOrchid社のFahrenheit1280を搭載している．価格は1,178,000円から．

アンテナハウス，DOS 5.0，Win3.0対応「MSKN Ver.3」を発売

アンテナハウス（株）（TEL 03-3234-9631）は，MS-DOS Ver.5.0のタスクスワップ機能に対応し，Windows 3.0/3.0A，またはマイクロソフト社の推奨するかな漢字変換仕様「MS-KANJI仕様」を満たすMS-DOSアプリケーション上でATOKによる日本語入力を可能にする「MSKN Ver.3」を発売

ワンラインNEWS
日本電気は13MBの大容量に対応，米アップルコンピュータのパソコンの方式にも使えるFDD用コントローラ「μPD72070」を開発した．

NEWS & NEW PRODUCTS

した．コマンドラインからファイル名を実行するだけでメモリに常駐し，常駐後は［CTRL］＋［XFER］キーでアプリケーションの中からATOKを利用することができる．またデバイス行での設定もサポートした．対応機種はPC-9800シリーズ，PC-286/386シリーズで，価格は9,800円．

エー・アイ・ソフト，日本語FEP「WXⅡ$^+$」をMS-DOS 5.0に対応

エー・アイ・ソフト（株）（TEL 03-3498-7301）は，日本語フロントエンドプロセッサ「WXⅡ$^+$」のPC-9801版をMS-DOS 5.0対応にリビジョンアップした「WXⅡ$^+$ 2.00F」，Windows 3.0の新バージョン，NEC Windows 3.0A，IBM Windows 3.02に対応した「WXⅡ-Win 1.1」を発売した．価格は両製品とも9,800円で，2月1日以降登録者カードを出したユーザーには無償で，そのほかのユーザーには手数料1,000円でディスクを発送する．また，現行バージョンとの差分をパソコン通信上で公開し，転載自由のデータとすることで各地パソコン通信ホストへの掲載を促す．すでにWXⅡ$^+$，WXⅡ-Winを購入しているユーザーは，それを入手することにより無料でリビジョンアップ，バージョンアップが行える．「WXⅡ$^+$ 2.00F」は，MS-DOS 5.0対応のほか，MAOIX対応，約10%の変換速度の向上と動作改善，MAPIの動作改善，オーロラエース辞書コンバータの追加などもなされた．「WXⅡ-Win 1.1」は，DOS/V IBM 101キーボードのサポート，起動キー設定の自由度の改善など，キーカスタマイズの改善，カスタマイズ環境の改善などの機能強化がなされた．

日本ワードパーフェクト，WordPerfect 5.1Jほかを発表

日本ワードパーフェクト（TEL 03-3780-0515）は，日本語ワードプロセッサソフト「WordPerfect 5.1J」を発表した．「WordPerfect」は全米で75%のシェアを誇るワープロソフトで，「WordPerfect 5.1J」は，欧文版「WordPerfect」をベースに日本語化したもので，日本語かな漢字変換プログラム「WPFEP Version1.0」を標準搭載している．また他社のかな漢字変換プログラムにも対応している．自動作表機能（表計算機能内蔵），数式編集エディタ（数式フォント内蔵），英単語スペルチェック（115,000単語），表計算ソフトとのリンクなどの機能をもち，スケーラブルフォントプリンタに対応．「WordPerfect 5.1」と完全なデータ互換を実現している．PC-9800シリーズ対応版は4月上旬，J-3100，PS/55版，DOS/V対応版は5月上旬の出荷予定．また同時にMacintosh用として「WordPerfect 2.1 for Macintosh」も発表，英語のシステム上では英文ワープロとして，漢字Talk上では日本語ワープロとして使用することができる．また，日本語禁則処理，ルビ機能などを言語モジュール（15,000円）で提供，日本語かな漢字変換プログラム「EGBridge version 5.1」をバンドルする．出荷は6月上旬の予定．価格は「WordPerfect 5.1J」，「WordPerfect 2.1 for Macintosh」ともに，標準パッケージが58,000円，追加ライセンスパッケージが44,000円（マニュアル付き），34,000円（マニュアルなし），20ユーザー追加ライセンスパッケージが580,000円（マニュアルなし）．

マイクロソフト社のMS-Networks，販売累計30万ノードを突破

マイクロソフト（株）（TEL 03-5484-0700）は，日本国内における「MS-Networks」の販売累計が1月末に30万ノードを突破したことを発表した．「MS-Networks」はPC-LAN用ネットワークオペレーティングシステムとして1984年から国内出荷を開始，出荷開始よりPC-LAN市場において75%のマーケットシェアを獲得した．MS-NetworksはOEM製品で，各社ハードウェアメーカー，LANベンダーからMS-Networks，Net/One，StarLAN，Net/GCなどの製品名で販売されている．

ワンラインNEWS
米シリコングラフィックスとミップス・コンピュータ・システムズが合併に合意したが，両社の日本法人の扱いについては未定となっている．

Programmer's Program

DDE対応スクリーンエディタ
FOXY for Windows Ver.1.1

1つのパッケージではほとんどのMS-DOSに対応するテキストエディタ「Final」を販売している㈱エー・エス・ピーが，低価格で軽快に動作するWindows 3.0対応の多機能エディタ「FOXY for Windows Ver.1.1」(以下FOXY)を発売した．β版を日本電気のPC-9801DS2，MS-DOS 5.0およびMS-Windows 3.0A上で試用する機会を得たのでレポートする．

FOXYの特徴

FOXYの特徴は以下の通り．

洗練されたGUI環境

リボンやコマンドボタンのサポートを行うことで，システムメニューだけでは得ることのできない，洗練されたマウスオペレーションを提供する．またWindows 3.1形式のダイアログボックスのサポート，キーボードマクロのアイコン化などビジュアルシェル環境にマッチした最新のGUI環境を実現した．

DDEサーバ機能を実現

Windowsにおける大きな特徴であるDDEサーバ機能をサポート．FOXYをほかのアプリケーションとリンクすることでサーバとなるFOXYで入力されたテキストデータは，リアルタイムにクライアント側のアプリケーションに転送される．この機能によりテキストの修正や変更がクライアント側のデータにも反映され，ワープロや表計算ソフトなどで再入力の無駄を軽減する．各種アプリケーションソフトとホットリンクで接続した状態となるため，テキスト入力におけるフロントプロセッサとして活用できる．

快適な操作を実現

FOXYは，より快適なテキスト編集をコンセプトに設計，使用する書体をシステムフォントに限定することで画面表示のCPU負担を軽減し，快適なフットワークを実現した．さらに，各ソフトキーに割り当てた64バイト×16個の文字列を使って，スピーディな日本語入力を行うこともできる．Windows環境におけるテキスト作成に必要な要素を追求することで，軽快な操作性を実現している．

Finalの操作性を継承

MS-DOS版エディタにおける操作の基本スタイルといえるファンクションキーへの対応や，WordStarライクなショートカットオペレーションの提供といったFinalの操作性を継承．FinalなどのMS-DOS版エディタと変わらないキーボードベースのテキスト入力を可能にした．

豊富な編集機能

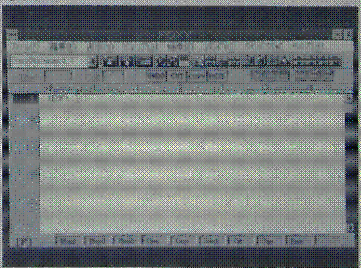
マルチファイル，カット&ペースト，ボックス型カット&ペースト，文字列の検索・置換，行のマークとジャンプ，フリーカーソル，ソフトウェアキーボード，ストリングスバンク，カスタマイズ，ビルトインgrep，タグジャンプなど，豊富な機能が使用できる．

ツールバーに機能ボタンを配置

Windowsのプログラムマネージャに登録されるFOXYのアイコンはキツネを現している．このアイコンをダブルクリックすることでFOXYが立ち上がる．もちろんWindows初期化ファイル「WIN.INI」の[Extensions]セクションでテキストファイルの拡張子を関連付けしておけば，ファイルマネージャから設定した拡張子をもつテキストファイルをダブルクリックするだけで，FOXYを立ち上げることもできる．

FOXYの画面はMicrosoft Word Version 1.2（以下MS-Word）やMicrosoft Excel Version 3.1（以下MS-Excel）などに似たデザインと

写真1



なっている．メニューバーの下にはリボン（ツールバー）と呼ばれる行があり，ここに各種コマンドボタンやコンボボックスがあり，その下にルーラーが配置される．また最下段にはファンクションキーメニューが表示されている．

リボンにあるコマンドボタンは，メニューバーから呼ぶよりも素早く呼び出すことができ，編集テキストのロード・セーブ，編集テキストの切り替え・選択，ファイルの検索・置換，カーソルのジャンプ，カット・コピー＆ペーストなどが用意されている．もちろん従来からのホットキーやショートカットキーもサポートしている．

ただPC-9801用のWindowsではスクリーンサイズが640×400であるため，複数のアプリケーションウィンドウを開くと個々のアプリケーションの表示範囲が小さくなり，いささか見づらい感がある．FOXYも例外ではなく，フルスクリーンの状態でテキストが表示できるのは68桁×15行（半角）で，ソースファイルを見るには，いささか辛いといえる．

MS-DOSエディタと同じ操作感

編集時のキーオペレーションはMS-DOS上でのFinalやMifesなどと同じで，ファンクションキーやショートカットキーを使ったオペレーションをそのまま使用することができる．キーカスタマイズ機能を使って，MS-DOS上で普段使い慣れたエディタとまったく同じキー操作を設定すれば，違和感なく使うことができるだろう．さらにカット＆ペーストやコピー＆ペーストなどの編集作業においては，マウスとリボン上のコマンドボタンを使って素早いオペレーションを可能にしてくれる．

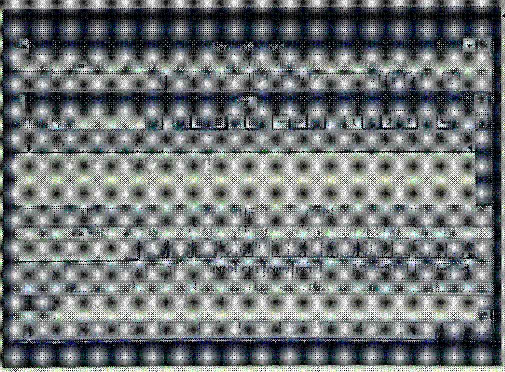
DDE機能を使ってテキストをリンクする

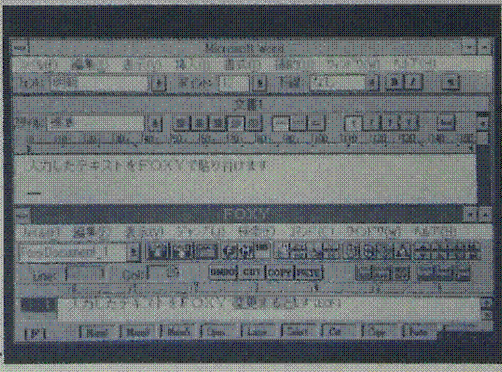
FOXYはDDEのサーバー機能を有している．FOXYで入力したテキストを修正すると，ホットリンクしたMS-ExcelやMS-Wordにリアルタイムに反映される．DDE機能を使うには，まずMS-Excelの場合は「他（O)」-「フルメニュー（M)」，MS-wordの場合は「表示（V)」-「フルメニュー（M)」であらかじめフルメニューに設定する必要がある．FOXYでリンクしたいテキストブロックを選択し，「編集（E)」-「複写（C)」で通常どおりにコピーする．次にExcel上でマウスまたはカーソルを使って張り込むセルを選択し，「編集（E)」-「連結貼り付け（L)」でペーストする．MS-Wordの場合も同様に，ペーストしたい位置にカーソルを移動し「編集（E)」-「拡張貼り付け（N)．．．」メニューから「リンクする（L)」または［RETURN］でペーストする．以後，FOXY上でテキストの変更があると，MS-WordやMS-Excel上のリンクされたテキストもリアルタイムに変更される（写真2，3）．

ほかにも短文登録機能やソフトウェアキーボードなど数々の特徴をFOXYはもっている．さらにFOXYの登録ユーザーに対してFOXYのソースコードが10,000円で配布される点も見逃せないだろう．FOXYの登場によってWindows上の開発環境に大きな変化が訪れるかもしれない．

■FOXY for Windows Ver.1.1

販売：（株）エー・エス・ピー（03-5493-7211）／価格：38,000円／適応機種：MS-Windows 3.0が動作する機種／動作環境：MS-Windows 3.0（スタンダードモードおよびエンハンスドモード）／供給メディア：5インチまたは3.5インチ2DD


◀写真２


写真３▶

オンラインソフト採集帳 17

多機能CDコマンド FCD
ディスク使用率の表示 gdf

多機能CDコマンド「FCD」

フリーソフトウェアのファイル管理ソフト「FD」(A. Idei作)を使っていると,「FDTREE. DAT」というファイルが生成されることがある. またEMSメモリ上に「FD_HIST$」というハンドル名をもつデータが確保される. FDTREE. DATは[SHIFT]+[L]または[T]によって表示されるツリー画面で[W]キーを入力すると生成される各ディレクトリのツリー構造のデータファイルだ. FD_HIST$には入力コマンド,[L](ディレクトリの変更),[C](ファイルのコピー),[M](ファイルの移動),[U](圧縮ファイルの復元)などで,指定したディレクトリなどの履歴が保存されている.

これらFDの生成するデータを活用した多機能CDコマンド,それが「FCD」である.

ビジュアルCD機能

MS-DOSのコマンドラインから

A> FCD B:

などと入力すれば,指定ドライブ(この場合はBドライブ,ドライブ名を省略した場合はカレントドライブ)のディレクトリツリーを表示する. これを見ながらカーソルでディレクトリを選択できる. また該当ドライブのルートディレクトリにFDTREE. DATがあれば,無数のディレクトリがあっても高速に表示することができる. ディレクトリの表示時に使用できるコマンドは[L](ツリー表示するドライブの変更),[↑]/[↓](カーソル移動,[SHIFT]キーを同時に押せば高速に移動),[<]/[>]/[?](先頭/最後/カレントディレクトリへカーソル移動),[W](ディレクトリ情報をFDTREE. DATに保存),[R](ディレクトリ再検索),[SHIFT]+[任意のキー](ディレクトリの頭文字検索移動)がある. [W]で読み出したディレクトリ構造がFDTREE. DATに保存される. また実際のディレクトリ構造と矛盾が生じたときは[R]で再検索を行い,正しいディレクトリを読み込ませる必要がある.

ヒストリ付きCD機能

MS-DOSのコマンドラインから

A> FCD TOOL

と入力すれば,MS-DOSの内部コマンドであるCD(CHDIR)同様にディレクトリの移動を行う. もちろんドライブ名を指定すれば同時にドライブの移動もできる. さらに存在しないディレクトリが指定された場合は,FD_HIST$を参照して指定されたディレクトリ名を検索し移動する. FD_HIST$がない場合,またはFD_HIST$でディレクトリが見つからない場合はツリー表示をして,前述のビジュアルCDモードとなる. ユニークなディレクトリ名であれば,ディレクトリの深さやドライブの違いを気にすることなく移動できる.

ディレクトリ履歴管理機能

A>FCD B: /W　…①
A>FCD /L　…②
A>FCD /L < DIR_HIST.TXT…③

①はディレクトリ構造を読み込み,ルートディレクトリにFDTREE. DATが作成される. これによってFDやFCDから高速にディレクトリ表示を行うことが可能になる. /Lオプションでディレクトリ履歴の表示・変更を行うことができる. たとえば②はFD_HIST$のディレクトリ履歴を標準出力に表示する. また③のように標準入力がリダイレクトされていると,ディレクトリリストを読み込み「FD_HIST$」のディレクトリヒストリへ登録する.

「ディレクトリを深くしすぎて同じディレクトリ移動が面倒」とか「たしかBドライブのどこかに作ったTEXTってディレクトリにファイルを入れたと思うんだけど」なんて悩んでいるズボラな貴方を応援するツールである.

ディスク使用率の表示「gdf」

「gdf」はディスクの使用率をグラフを用いて視覚的に表現するユーティリティである. 起動オプションは以下の通り. またこれらのオプションは環境変数GDFに指定することができる.

-a すべてのドライブを調査
-d グラフ表示はしない
-f FMR専用グラフ文字を使用
-g PC-9801専用グラフ文字を使用する

-i 内部フラグを初期化
-j 1バイトかな文字を使用
-k 1k＝1000とする
　(既定値 1024)
-l join / subst 情報のかわりにボリュームラベルを表示
-m モノクロで表示
-o 使用中でないドライブも表示
-s ボリュームラベルを小文字で表示（パス区切りは「/」）
-t 全調査ドライブのデータの合計を表示
-w グラフを幅広で表示
-x 使用量を表示
　(無指定時は残量表示)
-h ヘルプを表示
-v バージョンを表示

あらかじめgdfをパスの通ったディレクトリに用意し，環境変数GDFに「-aogtl」と設定して，MS-DOSのコマンドラインから

A> GDF

と入力すれば，すべてのドライブの使用状況を知ることができる．

これをAUTOEXEC.BATなどに加えれば，毎回起動時にドライブの使用率をビジュアルに確認することができる．そして，それぞれのドライブの表示が赤くなってきたら新しいハードディスクを導入するか，それとも使わなくなったファイルの整理をしてフリーエリアを空けるか，懐具合と話し合ってみるのに役立つだろう．

■FCD ver.1.23

名称：FCD／分類：ユーティリティ／登録名：FCD123.LZH／作者：くわじ(NIFTY-Serve NBG01623)／動作環境：PC-9801シリーズ／登録月日：1992年2月2日／復元方法：LHA E FCD123

■gdf ver.1.81

名称：gdf／分類：ユーティリティ／登録名：GDF181.LZH／作者：TAC-CHA (PC-VAN ZXC45278)／動作環境：MS-DOS 2.11以降（2.11では一部制限あり）／登録月日：1991年3月6日／復元方法：LHA E GDF181

ファイル構成

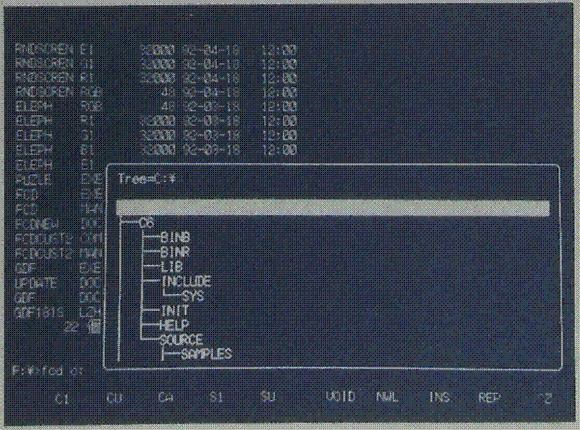
Listing of archive : FCD123.LZH

Name	Original	Packed	Ratio	Date	Time	Attr	Type	CRC
FCD.EXE	20513	13717	66.9%	92-02-02	01:23:00	a--w	-lh5-	371d
FCD.MAN	11258	4791	42.6%	92-02-02	01:23:00	a--w	-lh5-	A320
FCDNEW.DOC	4181	1783	42.6%	92-02-02	01:23:00	a--w	-lh5-	3319
FCDCUST2.COM	700	613	87.6%	91-08-01	01:14:00	a--w	-lh5-	6779
FCDCUST2.MAN	700	461	65.9%	91-08-01	01:14:00	a--w	-lh5-	3666
5 files	37352	21365	57.2%	92-02-02	01:23:00			

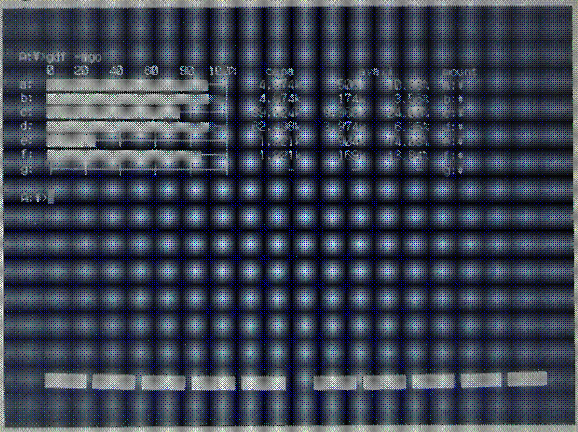
Listing of archive : GDF181.LZH

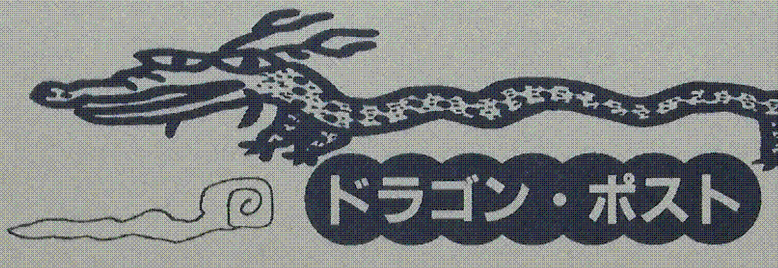
Name	Original	Packed	Ratio	Date	Time	Attr	Type	CRC
GDF.EXE	13174	8334	63.3%	91-03-06	01:39:00	a--w	-lh5-	FAF1
UPDATE.DOC	1888	929	49.2%	91-03-06	02:25:12	a--w	-lh5-	AA35
GDF.DOC	4959	2529	51.0%	91-03-06	02:32:46	a--w	-lh5-	D720
GDF181S.LZH	13459	13459	100.0%	91-03-06	02:32:38	a--w	-lh0-	0EFD
4 files	33480	25251	75.4%	91-03-28	01:49:12			

▼FCD



▼gdf





ドラゴン・ポスト

Illustration
Nobuki Fukuhara

この欄は，読者からのお便りを中心に構成します．本誌に対するご意見・ご感想などをお待ちしております．鬼より恐い読者からの声ということで，「ドラゴン・ポスト」です．本誌についているアンケートはがきにご意見の欄を設けてありますから，そこに書いてください．ペンネームでも歓迎です．ただし，ハガキには本名と住所をしっかり書いてくださいね．掲載誌をお届けします．

★ハンドヘルドを知ってるかい

ノートパソコンにカラー液晶だ!!　とさわがれている今，いったい何人の人が「ハンドヘルドコンピュータ」という言葉を知っているだろうか．PC-8201（NEC）やHC-20（EPSON）なんてのもあったんだけど……（そういえば，SHARPのPC3100ってどうなったんだろう．(時間蝿)

——ハンドヘルドコンピュータ!?　なんと懐かしい響きでしょう．ハンドヘルドコンピュータとカプラーをバッグに入れて旅行に行くのがトレンドだった時代もあったんですよねぇ．

★おじさんハッカー vs. せいねんハッカー

BookReviewの評価チャート，読者対象から定番度は理解できるが，ビョーキ度（懐かしい言葉ですね．編集部はYMO世代ですか？）で，“おじさん”と“ハッカー”が対極に位置づけられているのが，よくわからない．ぼくの知る範囲ではおじさんハッカーのビョーキ度は並のせいねんハッカーのビョーキ度をはるかにリョーガしているのだが．(竹田賢一)

——コンピュータの知識がまったくなく，かつ，いくら説明してもちっともわかってくれない人のことを“おじさん”と総称しているのです．ハッカーは年齢に関係なくハッカーでしょう．

★OS/2ユーザーの不安

世間のパソコンマニアの方々は，MS-DOSからDOS/V，MS-Windowsへ目が向いているようですが，当方，なにをまちがったかOS/2とUNIXで苦労しとります．OS/2はあと何年残るのやら，入れたのは誤りだったのかなあー．早めに全部UNIXに移そうかなあ．(P.P.)

——OS/2がどうなるかは，この春出荷されるOS/2 2.0のできいかんで決るでしょう．この春から年末にかけて，OS関係はけっこうおもしろい動きがあるのでは？

★新言語「PASIC」!?

昔は「BASIC」，そして今は「C言語」とプログラミング言語が移っていきますが，そんなにCは良いのでしょうか？　見やすさなら，構造化されたBASICの方が良いと思うし，デバッグも簡単なBASICのどこがいけないのでしょうか？　「遅い，オブジェクトが大きい，互換性が少ない」などとよくいわれますが，言語の仕様を改善してCより良い言語「PASIC」にしてしまえば良いと思います(基本はQuickBASICで，Cの#defineや漢字による変数名・サブルーチン名や固定文字変数に定義されていても，その文字数以下なら可変長文字列扱いなど，そしてインライン・アセンブラなど，その他いろいろ)．(二瓶努)

——こうして次々に新しい言語ができていくわけですね(笑)．

★クラッシュの悪夢

近頃，HDをクラッシュさせる夢をよく見る．家にはHDDはないのにおかしい．もしかすると，ゼミでこれから使うコンピュータのHDかも．この夢のおかげで，外部記憶装置へ出力するプログラムが，こわくて書けない．はやくこの悪夢から逃げ出したいものだ．(狂夢)

——思い切って１度HDをクラッシュさせてしまえば，もう恐いものはありません．きっと変な夢も見なくなりますよ．

★なかなかうまくいかなくて……

つい先日，うちの会社にロボットが来ました．(溶接の)操作などは本当にわかりやすいのですが，ソフトの方の開発がともなっていないため，こまかなセッティングが出来ないし，良いセッティングを見つけだすために，いろいろ数値を入力するのですが，なかなかうまくいかないのです．こんな時って，本当にゆううつ状態です（あーあ)．(羅夢哉)

——いっそROMを抜いて焼き直すってのはどうですか？いっきに解決できるような……バグが出ると地獄ですが．

★学生時代に得たもの

長い学生生活もあと１ヶ月ばかり．のんびりやったおかげで，コンピュータも少しはいじれるようになったし，こういうパソコン雑誌も読めるようになったから，ちょっと人生拡げられたかも知れない……．最近のディスク

の中身は286でHD無しの者には使えない物ばっかりで悲しいです．（伴秀利）
――以前にも書いたのですが，HDは必需品です．なにをおいてもまず買うべきです．

★マニアってどんな人？
(読者カードの）ユーザー区分の「マニア」って何？　もしかして……と思いあたる点はあるが（たしかに職場では「おたく」と私のことを呼ぶが），たまに大場久◯子の歌を聞き（もちろんレコード），パソコンの画面では某アイドル（り◯ちゃんではない！）が，ほほえんでいるけど，衛星アニメげきじょうを見て島崎◯子（ROKO）もがんばっているなあアーと思っているが，「珠里亜」の会員番号が「18-78002」だったけど……違うと思う．それに，夜中おきてプログラムらしきものを組むのは普通の行動だから，当然違う．「マニア」ってどんな人なのでしょうか．（うらしまん）
――マニアというのは「熱狂者」のことですから，ようするにパソコンを熱狂的に好きだと自覚している人のことです．うらしまんさんは，パソコンマニアというよりは……

★ハード増設にかける呪文
さあ卒業だ就職だ．バリバリ増設するぞ．メインメモリを6M以上にして，ハードディスクも２台目を買って，98シリーズもFAがでたのだから，私のRAにもTURBO 486 DXをつけて，レーザープリンタも買って，CG用フレームバッファも買って……，ゲロゲロ予算が大オーバー，欲しいソフトも多いのに，なぜにハード増設だけでこんなに高価になるのでしょうか．私にできる最大のことは呪文をかけることだけだ．「安くなれ，安くなれ!　安くなれ!!」（SEI）
――いまもっている98と周辺機器，ソフトなどを全部売っぱらって，IBM PC互換機に乗り換えればずっと安くていい環境が手に入りますけど．

★8086アセンブラはややこしい
ついにPC-9801ns/Eを買った．届いた数日後にns/Tが発売された．ほぼ予想通りの性能だったので，それほどショックは受けなかった．よし，これでプログラミングに専念できる．BORLAND C++もインストールしたし．それにしても8086アセンブラってややこしいですね．「Z-80がわかっているからだいじょうぶ」とタカをくくっていたのだが，やはりアマかった．さて，いつになったら追いつくのやら……．（Ship）
――MASM 6.0はまるでC言語みたいになっちゃったし，TASMも新しいものはオブジェクト指向アセンブラになるとかいうし，アセンブラはいったいどうなっちゃうんでしょう？

★PASCAL指向の雑誌になって
巷ではやりのC言語，初心者向けの解説本も多数あり，私もご他聞にもれず，TURBO Cを持っているのですが，初心者にはPASCALの方がいいように思います．そこで，貴誌には是非PASCALの方へと指向してもらいたいのです．CやBASICの雑誌はいろいろあるのですが，PASCAL専門誌が何故か無い．そしてPASCALファンを増やしましょう．（真剣にTURBO PASCAL 6.0の購入を考えている“あにま”でした）
――Pascalの雑誌といえば，アメリカ(?)に「Pascal Ada Modula-2」とかいうのがありましたねぇ，最近なぜか見かけませんけど．

★もっと基礎的なことを……
昔に比べ，PCのパフォーマンスを考えるとだいぶ値段も下がりました．今のスタンダードは５年前のサーバ用のポテンシャルを持っています．職場の中の人を見ていると，何と無駄な使い方をしているのかと，頭を抱えることも何度かあります．OAが当たり前でやっている訳ですが，もっと基礎的なことを企業として教育すべきと考えています．私の手持ちのPCも“頭”がだいぶ弱くなってきました．'93年には486/50MHz購入，目指して車も車検に出してお金を貯めたいと思っています．
――93年なら586マシンが出ているんじゃないですか？486なんてゴミだぜとかいわれてたりして……

★なでなでしたら治った
先日，私のパソコンが暴走した．プログラム言語（アセンブラ）の都合上，暴走は日常茶飯事なので，「よくあることさ」と，リセットを押した直後，私は硬直した．な，なんと，FDDのドライブＡが使用不能になっているじゃありませんか!　イニシャライズ時にFDDのＡドライブではなく，Ｂドライブを読んでしまう!　これはこまった．電源を切っても，やはり同様にＢドライブを読んでしまう．なんでだろう……うーん，「そうか最近使いすぎているから，つかれているんだ．ごめんよー．なでなで．」としてから，電源を入れたら元にもどった．私のパソコンにはグレムリンが棲んでいる．（橋本秀俊）
――２～３日徹夜でデバッグしたことのあるプログラマならば誰でも，どこからともなく現れてデバッグを手伝ってくれる小人さんのことを知っているでしょう．でも，もしグレムリンなら夜12時をすぎたらエサをあげてはいけません（笑）．

（敬称略）

SE

付録ディスクのご案内

このフロッピーディスクに含まれる実行ファイル，ソースリストおよびテキストファイルは「月刊Programmer's Page」1992年5月号に掲載されたものです．

付録のフロッピーディスクは，5"2HD（8セクタ／トラックMS-DOS）でフォーマットされています．3.5"や2DDなどへのコンバートサービスは行っておりません．なお，オンラインサービスSE-NET（03-5276-1297）でも，同じ内容のファイルを用意しています．

フロッピーディスクの内容は次の通りです．本誌およびディスクに含まれるドキュメントファイルをよく読んでからご利用ください．

■ディスクのおもな内容

●付録ディスクについての説明

ディレクトリ名：ルート

収録ファイル名：

README.BAT（ディスク内容の説明を読むためのバッチファイル，PC-9801専用）

README.DOC（ディスク内容の説明，MS-DOSテキストファイル）

ONLINE.BAT（オンラインソフトの説明を読むためのバッチファイル，PC-9801専用）

ONLINE.DOC（収録されているオンラインソフトの簡単な説明）

V.EXE（ミニページャー，PC-9801専用）

BOOK.TXT（翔泳社書籍目録）

●特集「Cの学び方・教え方」

②「Cの基本関数をマスターする」（p.15）

ディレクトリ名：TOKU

収録ファイル名：TOKU2.LZH

⑤「2大C言語処理系互換ライブラリの作成」（p.40）

ディレクトリ名：TOKU

圧縮ファイル名：TOKU5.LZH

●実践C塾　アルゴリズム篇（p.71）

ツリー・その1

ディレクトリ名：JUKU

収録ファイル名：JUKU.LZH

●プログラム「遊」工房（p.86）

ゲーム「DYNAMO」

ディレクトリ名：PLAY

収録ファイル名：PLAY.LZH

●イカすBASICプログラミング入門（p.92）

実戦的リフテクニック編

ディレクトリ名：BASIC

収録ファイル名：BASIC.LZH

●98活用アセンブラ技法　ブロックデバイス編（p.102）

DOS BIOSの置き換え

ディレクトリ名：98ASM

収録ファイル名：ASYNC.ASM

●綴じ込み付録

ディレクトリ名：DATA

収録ファイル名：MSDOS50.TXT（MS-DOSテキストファイル）

●オンラインソフト採集帳（p.136）

今月号で紹介した多機能CDコマンド「FCD」および，ディスク使用率表示ツール「gdf」と，1つまたは複数のファイルの圧縮および「.LZH」という拡張子をもった圧縮ファイルの復元をする高圧縮復元書庫管理ソフト「LHA」が収録されています．なおFCDおよびgdfはLHAによって，LHAは自己復元型で，それぞれ圧縮されています．復元方法についてはONLINE.DOCを参照してください．

ディレクトリ名：ONLINE

収録ファイル名：

GDF181.LZH（gdf v.1.81，MS-DOS 2.11/3.xx用）

FCD123.LZH（FCD v.1.23，PC-9801用）

LHA213.EXE（LHA v.2.13，MS-DOS汎用）

●クイズ・ランダムスクリーン（p.150）

クイズ解読ツール「PUZLE」

本誌掲載のクイズ・ランダムスクリーンの画像をディスプレイ上に再現し，マウスを使って並べ換えることにより，解答を得ることができるユーティリティです．PC-9801シリーズ専用です，詳しい使い方はPUZLE.DOCを参照してください．

ディレクトリ名：OMAKE

収録ファイル名：

PUZLE.EXE（解読ツール，PC-9801専用）

PUZLE.DOC（PUZLEの説明）

CONFIG.SYS（PUZLE用CONFIG.SYS）

AUTOEXEC.BAT（PUZLE用AUTOEXEC.BAT）

RNDSCREN.LZH（画像データ）

ELEPH.LZH（4月号の解答データ）

ELEPH.BMP（Windows 3.0の壁紙）

●そのほか

カノープス電子（株）の協力によりJPEG形式の画像データを表示するJPEGローダー（サンプル版）のPC-9801版およびX68000版を収録しています．PC9801というディレクトリはPC-9801，X68000というディレクトリにはX68000版が収録されています．

ディレクトリ名：JPGV¥PC9801

収録ファイル名：

README.DOC（簡易ドキュメント）

JSMP.TXT（カノープスからのお知らせ）

JPGV.EXE（JPEGローダ，PC-9801用）

GSET.EXE（PC-9801のグラフィックVRAMおよびパレットの操作）

FROMMAC.EXE（Macintosh用JPEGファイルをPC-9801（MS-DOS）用に変換する）

SAMPLE.JPG（サンプル画像）

ディレクトリ名：JPGV¥X68000

収録ファイル名：

README.DOC（簡易ドキュメント）

JSMP.TXT（カノープスからのお知らせ）

JPGL.X（JPEGローダ，X68000用）

JPGS.X（X68000のグラフィックVRAMに表示した画像をJPEG形式でセーブ）

FROMMAC.X（Macintosh用JPEGファイルをX68000用に変換する）

SAMPLE.JPG（サンプル画像）

※使用上の注意

本誌に掲載されたプログラムおよびデータを利用するには，MS-DOSの基礎知識が必要です．このディスクには，MS-DOSのシステムは入っていません．別途，MS-DOSが必要となります．また，ソースファイルのコンパイルなど，別途コンパイラやユーティリティが必要な場合があります．

■圧縮ファイルについて

ディスクスペースを有効に使うために，多くのファイルは圧縮した上で収録しています．「.LZH」という拡張子のついたファイルは，高圧縮書庫管理ユーティリティ「LHA」によって圧縮されたファイルです．これらのファイルを利用するには，まずLHAで復元する必要があります．LHAの利用方法については，ONLINE.DOCをご覧ください．

バーチャルスペースガイド

P子　某ソフトハウスで企画営業を担当している．26歳．
C平　某メーカーのシステムソフトウェア開発部門に所属．25歳．

P子　どうしたのよ．うかない顔しちゃって？
C平　うん．まあ．
P子　何なのよ．ぐずぐずして．
C平　昨日，家に帰ってゆっくり遊ぶ暇がなかったんだけど，昨日は久しぶりに早めに帰って，前から見ようと思っていたレーザーディスクを見たのよ．それが，どうも面白くないんだ．昔見たときの感じと違う．
P子　また，何べんも同じ映画見て，オタクしているのね．で，何なのそのLDは？
C平　ディズニーの「ファンタジア」．子どものときに見たときは，感激したものだったけど，もう集中して見ていることができなくなっちゃって……．面白くないんだよね．疲れているのかな？
P子　あ，あれなら私も小さいときにみたよ．そんなに面白いとは思わなかったんだけど，きれいな映画だな，という記憶がある．
C平　そうなんだ．いま見てもきれいだとは思う．あれが，1940年に作られたということもすごいとは思う．でも，頭でそう思うだけで，集中して見ることができない．やっぱり，疲れているのかな．
P子　歳とっただけじゃない．それに映画って感激するにはタイミングがあるのよ．
C平　うーん．そうかもしれない．アメリカで公開された当時は酷評されたらしいけど，60年代後半のサイケデリック時代に評価されるようになったというから……．

■ファンタジア

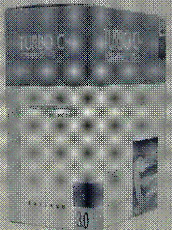
(1940年，米)
ウォルト・ディズニー作品
レオポルド・ストコフスキー指揮
フィラデルフィア管弦楽団
7,300円(税別)　パイオニアLDC

ムソルグスキーの「禿山の一夜」，チャイコフスキーの「くるみ割り人形」，ベートーベンの「田園」，モーツァルトの「アヴェ・マリア」など名曲のイメージをアニメーションで表現した音楽アニメの古典的作品．片面最大7分のSPレコードの時代に，サウンドトラックで9チャンネル超ステレオを実現していた．ビデオ，LDはドルビーサラウンド．



読者プレゼント

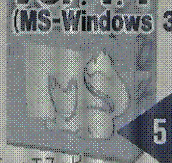
1　Turbo C++ for Windows 3.0 (IBM PC用)



提供：ボーランド(株)

3名

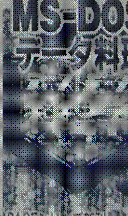
2　FOXY for Windows Ver.1.1* (MS-Windows 3.0対応)



提供：(株)エー・エス・ピー

5名

3　MS-DOSテキストデータ料理学



翔泳社・PC-PAGEシリーズ新刊

3名

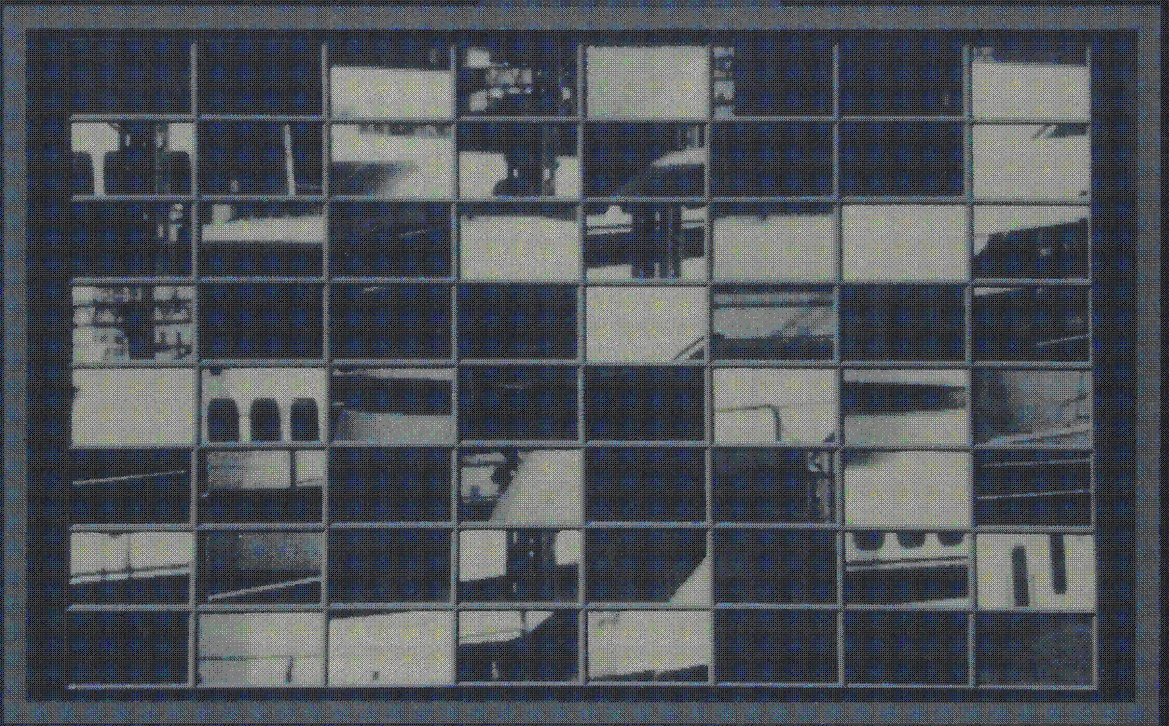
応募は本誌に添付のアンケートはがきをご利用ください．(1992年5月18日消印有効)．なお，応募者多数の場合は，抽選になります．当選者発表は発送をもって代えさせていただきます．

＊FOXY for Windows Ver.1.1に当選された方には，一定期間の使用の後，製品に関するレポートを提出していただきます．

クイズ・ランダムスクリーン

ある１枚の画像を64分割し，ランダムに並べ替えてあります．さて，もとの画像は何を写したものでしょうか．

ヒント：ひかりより速い



正解者の中から30名様に本誌表紙イラスト（日暮修一画）入り月刊プログラマーズ・ページ特製オリジナルマグカップをさしあげます．

応募方法：本誌に添付のアンケートはがきをご利用ください．

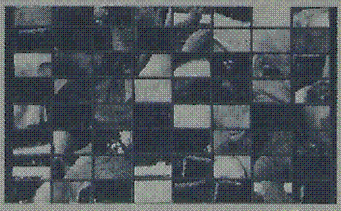
締め切り：1992年５月18日（消印有効）

なお，当選者の発表は発送をもって代えさせていただきます．

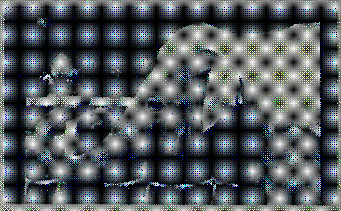
＊画像データは，付録ディスクやBBSでもサービスしています．

４月号の問題と正解

【問題】



【正解：ぞう】



SE-NETからのお知らせ

SE-NETでは開設以来，オンラインサービスを行ってきましたが，回線混雑のため非常に接続しにくい等の問題が出てまいりましたので，GUESTによるダウンロードは中止させていただきます．今後は読者のみなさまと編集部とのコミュニケーションの場としてご利用ください．

●BBSへの接続

SE-NET　03-5276-1297

通信速度	300/1200/2400bps
接続プロトコル	CCITT V.22bis/V.22/V.21
エラー訂正プロトコル	CCITT V.42，MNPクラス５
漢字コード	シフトJIS

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ6月号

(5月18日発売)

付録：5″ 2HDフロッピーディスク付き!!

特集

アセンブラの作り方

8086系アセンブラの設計と作成

新しく開発されたCPUを利用する際など，プログラマがアセンブラ処理系を作成しなければならないことも少なくない．この特集では，MS-DOSと8086系CPUを題材に具体的にC言語によるソースコードを示しながら，アセンブラ処理系の作成方法について紹介する．

月刊プログラマーズ・ページ　1992年5月号
1992年5月1日発行
発行所　株式会社翔泳社
東京都千代田区平河町2-4-14平河町KSビル4F・5F
TEL 03-3263-0447　〒102
発行人　長廻健太郎
編集長　清水　隆
副編集長　鈴木嘉平
編集スタッフ　杉野貴美子＋上田純美礼＋芳山充弘
表紙デザイン　SE（翔泳社デザイン研究所）
広告スタッフ　相澤正夫＋澤本良知＋種村嘉彦
協力　根本裕三郎＋(株)ドモン・マインズ＋保坂庸介
＋小林弘人(図版デザイン)
写植　株式会社東巧社＋株式会社トーヨー・プリント社
印刷　凸版印刷株式会社
定価　980円（本体951円，税29円）

EDITOR'S PAGE

●5月号が出たあと，6月号を発行するまでの間に，小社の事務所は移転する予定だ．2年間慣れ親しんだ平河町から西麻布にかようことなる．その前に4月から組織が改編され本誌の編集部も少し陣容が変わることになるだろう．現在，プログラマーズ・ページをベースにさまざまな企画を練っているところなので，ご期待いただきたい．(TA)

●最近のボーランドのバージョンアップは，すさまじいとしかいいようのないペースで行われている．ついこの間Borland C＋＋を2.0＋Application Frameworksにバージョンアップしたと思ったら，もう3.0へのバージョンアップが始まるというし，アメリカではWindows 3.1が発売された1か月後には3.1対応のものへとまたバージョンアップするそうだ．はっきりいって，マニュアルを読み終る暇もない．新しいMS-C/C＋＋ 7.0にも膨大なマニュアルがついてくるというし，ああ，頭が痛い．(嘉平)

●先日，不動産やさんからアパートの契約更新の手紙が届き，開けてビックリ！　なんと一挙に4000円も家賃が値上がり，2年前の更新ではたしか2000円の値上げだったのに，悔しいから引っ越そうかと思い，住宅雑誌で検討したけど，どこも高いんですね．あきらめて素直に更新します．(ま)

●プログラマの仕事場でお邪魔した日本ワードパーフェクトのセミナールームは，本来はビルのオーナーの音楽練習室で，女優の浅野温子がドラマ「101回目のプロポーズ」のためにチェロの練習をした部屋でもあるそうです．代官山という場所柄か，とても外からではソフト会社のオフィスがあるビルには見えませんでした．ところで春は異動の季節，私も今月限りでプログラマーズ・ページの編集部を離れることとなりました．次はPC-PAGEシリーズでお目にかかりましょう．(kos)

●プロフィールプロを入手したので，これを機会に衛星放送に手を出すことにした．で，買ってきたのはBSチューナー付きのビデオデッキEV-BS3000だった．今はせっかくのHi8だから，それを生かせるソースが欲しいと思っている．きっと秋葉原でレーザーディスクプレイヤーを今週末にでも買ってしまうのだろう．なににつけても金がかかる．(yoshi)

愛読者カードご記入のお願い

本誌をお買い上げいただき、誠にありがとうございます。
お手数ですが、今後の誌面づくりの参考にさせていただきたいと存じますので、
各欄にご記入の上、ご返送くださるようお願い申しあげます。

月刊プログラマーズ・ページ READER'S SURVEY CARD 愛読者カード

本誌の号数　　年　　月号

本誌を知った理由
- □1. 書店店頭　□2.他社出版物(　　　)
- □3.小社出版物　□4. DM　□5.口コミ

本誌を購入した書店　　市(区)　　書店

パソコンの所有形態(機種名)
- □1.個人所有　(　　　)
- □2.職場にある(　　　)
- □3.持っていない

よく使用するソフトウェア

よく使用するOS

今月号でおもしろかった記事(いくつでも)

今後取り上げてもらいたい特集のテーマ

貴方が選ぶベスト3

パソコン雑誌	①	②	③
ソフトウェア	①	②	③
パソコン本体	①	②	③

本誌についてのご感想、お気づきの点、その他どのようなことでも結構ですので、ご意見をお聞かせください(「ドラゴン・ポスト＝読者の声」欄に掲載させていただく場合もあります。ペンネームをご希望の方はあわせてご記入ください)。

月刊プログラマーズ・ページ 広告資料請求カード

8	会社(学校)名		9	所属(学部)		10	従業員数	

※下記の該当する番号をご記入ください。

11	業種		12	職種		13	パソコン利用状況	

業種　①製造業　②金融・保険・不動産業　③商社・卸・小売業　④建設業　⑤通信・運輸　⑥システム・ソフト　⑦その他サービス業　⑧公務員　⑨その他

職種　①コンピュータ(ハード)技術者　②コンピュータ(ソフト)技術者　③エレクトロニクス技術者　④プログラマ　⑤その他の技術者　⑥専門職　⑦事務・営業職　⑧経営管理職　⑨商工サービス　⑩教職　⑪学生　⑫その他

パソコン利用状況　①会社で利用　②個人で利用　③会社でも個人でも利用　④利用していない

●希望される広告資料　　年　　月号

希望項目 ＼ 広告資料請求No.												
A 購入希望	A	A	A	A	A	A	A	A	A	A	A	A
B 説明希望	B	B	B	B	B	B	B	B	B	B	B	B
C 資料希望	C	C	C	C	C	C	C	C	C	C	C	C

定期購読のご案内

綴じ込みの振替用紙を使って、本誌の定期購読をお申し込みになれます。
1年間12冊＝11,760円（本体11,417円/税343円）
郵送料サービス

郵便はがき

料金受取人払
麹町局承認
3908
差出有効期間
平成4年11月
30日まで
切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル
株式会社 翔泳社
マーケティング部 行

月刊プログラマーズ・ページ READER'S SURVEY CARD 愛読者カード

氏名	性別 □1男 □2女	生年月日 19 年 月 日
住所 〒		電話 ( ) －

職業
□1ソフト技術者 □2ハード技術者 □3技術系会社員
□4事務系会社員 □5営業系会社員 □6教師
□7一般公務員 □8経営管理職 □9理系学生
□10文系学生 □11その他（ ）

ユーザー区分
□1 プロ
□2 マニア
□3 一般ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

郵便はがき

料金受取人払
麹町局承認
3910
差出有効期間
平成4年11月
30日まで
切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル
株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

＊当社で記入

1	SE	2	受付No.

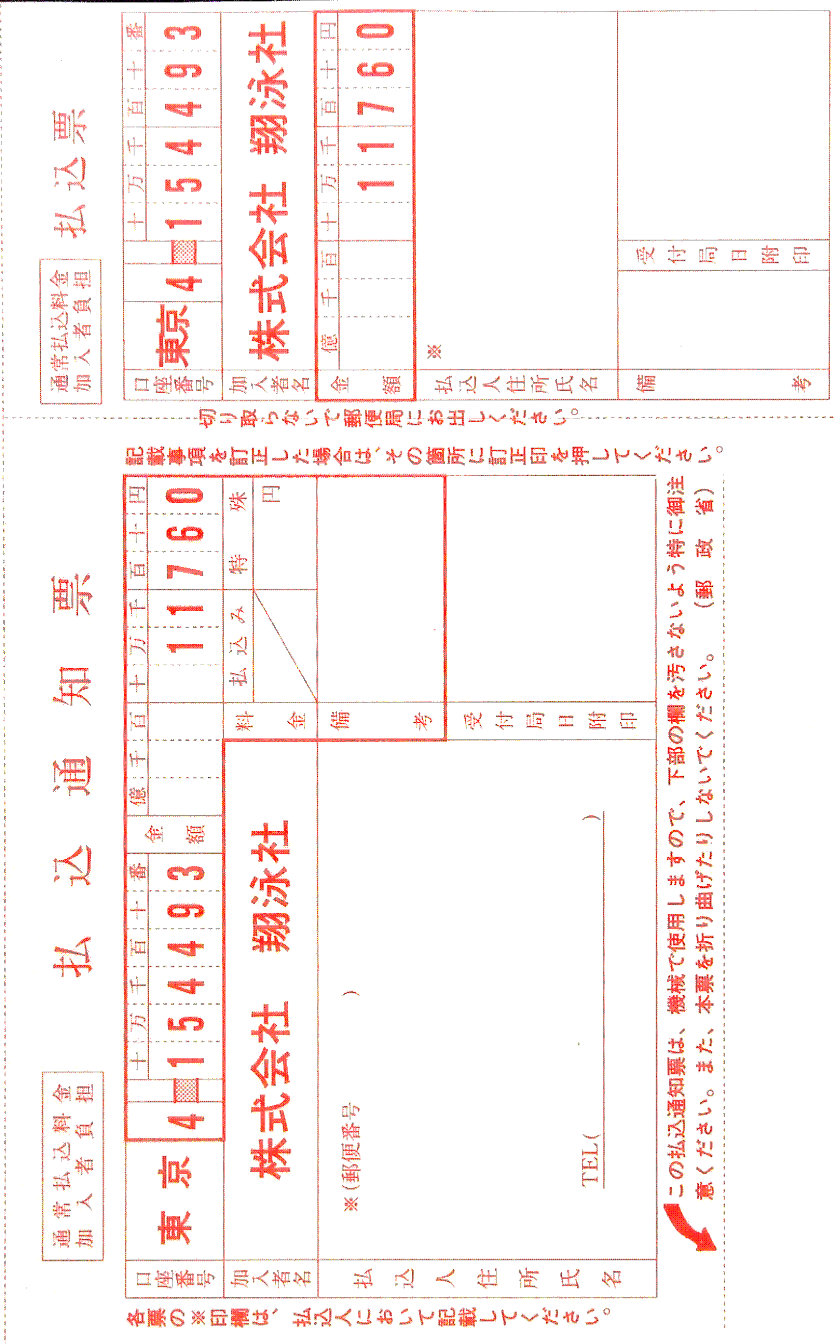
月刊プログラマーズ・ページ広告資料請求カード

3	会員番号	

＊すでに会員番号をお持ちの方はご記入ください。

4	フリガナ 氏名	姓 名	5	生年月日	19 年 月 日
6	自宅住所	〒 電話 ( )			
7	会社住所	〒 電話 ( )			

資料送付希望先の番号に○印をつけてください。


払込票
通常払込料金加入者負担
口座番号 東京 4-154493
加入者名 株式会社 翔泳社
金額 11760
※
払込人住所氏名
備考
受付局日附印
切り取らないで郵便局にお出しください。
記載事項を訂正した場合は、その箇所に訂正印を押してください。
払込通知票
通常払込料金加入者負担
口座番号 東京 4-154493
金額 11760
加入者名 株式会社 翔泳社
料金
払込み
特殊
円
備考
受付局日附印
払込人住所氏名
※（郵便番号　　　）
TEL（　　　）
この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。（郵政省）
各票の※印欄は、払込人において記載してください。

…ス株式会社

…いてきたLANビジネスの世…Nの先進国・米国で磨いた技…5年にして世界でも有数の総…たアライドテレシス(株)の存…

…取締役社長の松原由高氏を訪…自の企業理念などについてお…

きな夢を描くようになった。その夢を実現するために設立されたのがアライドテレシスであり、その目標は〝LANの世界ナンバーワン企業〟になることである。

「LANで最も大切なのはマルチベンダーであり、オープンアーキテクチャーの考え方であると思ってきましたから、企業自体も自由で独立した存在であるべきだと考えました。そこで、特定のスポンサーを持たない独立系企業として出発したわけです。そのために

こうというロマンですね。

このマーケットづくりと企業づくりの2つの面で、われわれの独自性を発揮していきたいと考えています。」

LANの夢実現に向け 日本発、世界へ展開

同社は〝世界ナンバーワン〟をめざすだけあって、海外法人の設立にも意欲的だ。すでに米国、英国、ドイツ、シンガポールに法人を設立、近くフランスへの進出も予定されている。

…ガマン・バス時代からの友人・…アメリカをはじめ、ヨーロッパ…アにたくさんいましたから、人…トワークを生かしていこうとい…は当初から考えていました。マ…ト情報、技術情報、生産資源と…ものは世界中にあります。それ…ーバルな資源をフルに活用する…のが、われわれの戦略なんです。…法人については、初期段階での…人以外は現地スタッフの経営戦…べてを任せて、可能な限り早い…資本面でも現地法人主体の企業…敷けるようにしています。親会…会社というより〝ブラザー・カ…ー〟的な関係で発展させていき…思っているんですよ。先日の会議でも、それぞれがその国で一番のLANベンダーをめざそう、その総和がグループとして世界一のLANベンダーになることだ、と話してきました。」

では、今後LANベンダーとしてトップに立ち、世界のLANビジネスをリードしていくために、同社はどのような方針を打ち出しているのだろうか。

「いま述べた国際戦略がその1つ。グローバルな資源を活用するということです。各国に展開しているリソースを連合・統合し、優れた技術開発力をベ

定期購読のご案内

■定期購読は、1年間12冊と限定させていただきます。

■購読料のお支払は、この振替用紙をご利用ください。銀行振込、現金書留によるご送金は、ご遠慮ください。

■振替用紙に、ご住所、お名前をご記入のうえ、郵便局の貯金窓口にてお支払ください。

■小社より領収書の発行はいたしませんので、郵便局にて「郵便振替払込金受領証」をお受取ください。

■お申し込みいただいた方には、弊社より直接発送いたしますので、到着が発売日より遅れることがございますが、ご了承ください。

この欄は、加入者あての通信にお使いください。

月刊プログラマーズ・ページ　定期購読直接申し込み書

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

年　　月号より

年間購読料 11,760 円（税・送料込み）

■ご希望の購読開始月号をご記入ください。

■送付先（住所・氏名は正確にご記入ください）

住所	□□□-□□	フリガナ	
		氏名	
		TEL	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。（郵政省）

郵便はがき

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社

月刊プログラマーズ・ページ
広告資料請求係行

2	受付No.	

マーズ・ページ広告資料請求カード

＊すでに会員番号をお持ちの方はご記入ください。

名	5	生年月日	19　年　月　日

□10文系学生　□11その他（　　　）　ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

6	自宅住所	〒　電話（　　）	資料送付希望先の番号に○印をつけてください。
7	会社住所	〒　電話（　　）	

綴じ込みの

アライドテレシス株式会社

ちょっと気になるこの会社

このところにわかに活気づいてきたLANビジネスの世界だが、そのなかでも、LANの先進国・米国で磨いた技術と経験を生かし、設立から5年にして世界でも有数の総合LANベンダーへと成長したアライドテレシス(株)の存在は、キラリと光っている。

東京・品川区の本社に 代表取締役社長の松原由高氏を訪ね、同社の事業展開やその独自の企業理念などについてお話をうかがった。

マーケットづくりと企業づくりと

アライドテレシス(株)は、1987年の設立以来わずか5年の間に、イーサネットを中心LANボードなどを供給する有力なLAN総合ベンダーとして急成長してきた、業界注目のベンチャー企業だ。

社長の松原由高氏は、日本電気でロケットの追跡・制御や衛星の管制など宇宙関連システムの関発を手掛けたのち、米国のアンガマン・バス社に入社。主任技師、研究開発部長、営業開発部長などを歴任し、LANマーケット開拓やLAN機器・システムのマネジメントを担当するなかで、LANの将来に大きな夢を描くようになった。その夢を実現するために設立されたのがアライドテレシスであり、その目標は〝LANの世界ナンバーワン企業〞になることである。

「LANで最も大切なのはマルチベンダーであり、オープンアーキテクチャーの考え方であると思ってきましたから、企業自体も自由で独立した存在であるべきだと考えました。そこで、特定のスポンサーを持たない独立系企業として出発したわけです。そのために初めはたいへん苦労もしましたが、やはり、しっかりした理念、姿勢をもつことが企業にとって一番大切なんじゃないでしょうか。

まず、われわれにはLANへの思い入れが強くありましたから、LANを普及させていくことでエンドユーザーはきっとハッピーになれる、それによって当社は社会的にも貢献できるんだと、そういう思い入れを大事にしてきましたね。

また、企業づくりにもチャレンジしようじゃないかということで、研究開発型の、いわゆるベンチャー企業として、これまでにない新しいスタイルを作り上げたいと思ってます。私とともに終わってしまう企業ではなく、恒久的に存続していく企業を作り上げていこうというロマンですね。

このマーケットづくりと企業づくりの2つの面で、われわれの独自性を発揮していきたいと考えています。」

LANの夢実現に向け 日本発、世界へ展開

同社は〝世界ナンバーワン〞をめざすだけあって、海外法人の設立にも意欲的だ。すでに米国、英国、ドイツ、シンガポールに法人を設立、近くフランスへの進出も予定されている。

「アンガマン・バス時代からの友人・知人がアメリカをはじめ、ヨーロッパやアジアにたくさんいましたから、人のネットワークを生かしていこうということは当初から考えていました。マーケット情報、技術情報、生産資源といったものは世界中にあります。それらグローバルな資源をフルに活用するというのが、われわれの戦略なんです。

海外法人については、初期段階での資本投入以外は現地スタッフの経営戦略にすべてを任せて、可能な限り早い時期に資本面でも現地法人主体の企業体制を敷けるようにしています。親会社・子会社というより〝ブラザー・カンパニー〞的な関係で発展させていきたいと思っているんですよ。先日の会議でも、それぞれがその国で一番のLANベンダーをめざそう、その総和がグループとして世界一のLANベンダーになることだ、と話してきました。」

では、今後LANベンダーとしてトップに立ち、世界のLANビジネスをリードしていくために、同社はどのような方針を打ち出しているのだろうか。

「いま述べた国際戦略がその1つ。グローバルな資源を活用するということです。各国に展開しているリソースを連合・統合し、優れた技術開発力をベ

ースに通信分野における新しいマーケットを創り出しながら、グループ全体を発展させていきたいと思っています。

2つ目には、マーケティング主導型のベンチャービジネスを開拓していくことです。マーケットで何がもとめられているのかということを常に考え、行動する企業であることを、われわれの強みにしていきたい。ここに技術があるから、それを製品にしようという発想ではなしに、ユーザーが何を求めているかを知り、それを実現するためにわれわれの力を集めようという、ユーザーニーズ優先の製品開発を進めたいと考えています。

3つ目は、われわれ自身が常にチャレンジ精神をもちつづけることです。ユーザーニーズに本当に応える製品を作ろう、これまで市場になかった新しい製品を生みだそう、そういうチャレンジを100%の目標で満足せず、120%の目標をもって取り組む姿勢が大切ですね。

そして4つ目には、バランスのとれた組織づくりを心がけています。企業はモノ、ヒト、カネだと言われますが、それに〝ユーザーからの信頼〟を加えた4つのバランスをとりながら、会社を発展させていくことが目標です。」

ユーザー本位に徹した LANの製品開発

さて、ようやく本格化しはじめた日本国内のLANビジネスだが、このなかで、アライドテレシスはどのようなポジションをとっていくのだろうか。総合LANベンダーとしての同社の役割について、松原社長はこう語る。



代表取締役社長 松原由高氏

「アライドテレシスは、あくまでメーカーだと考えています。製品そのものは委託製造という形でやっていますから、われわれ自身が工場を持って製品を生産することはしていません。当社のもつエンジニアリングの知的財産を、製品化し、アライドテレシスのブランドで提供するということです。

そのなかで一貫して大切にしているのは、ユーザー本位の姿勢です。LANインターフェース・カードやトランシーバーなど、これまでに数多くの高品質・低コストを心がけた製品を提供させていただいてきましたが、それらがユーザーの方々に支持されているのは、製品の開発・設計からサポート体制まで、ユーザー本位の姿勢を貫いてきたからだと思っています。

このところ、景気が後退してきたことによって、ちょっと上滑りだったLANのかけ声も沈静化してきたようです。これでかえってユーザー側のLANに対する考え方も地に足がついてきたんじゃないでしょうか。つまりいいものしか買わなくなってきた。その中でアライドテレシスの製品がいい評価をいただきはじめているということはうれしいことですし、手応えを感じています。今後も〝LANを入れないと乗り遅れる〟といった強迫観念をユーザーに植え付けるのではなく、LANで何ができるかという積極的な提案をしていきたいですね。」

人材主義が 企業の柱

最後に〝新しいスタイルのベンチャー企業〟をめざすという同社の、人材に対する考え方をうかがってみよう。

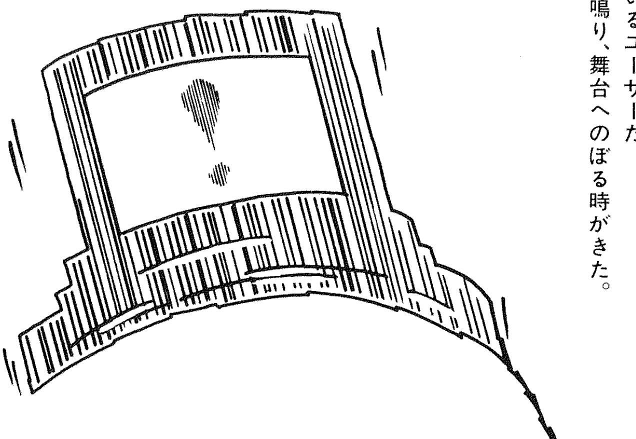
「人材配置としては、技術者集団にかたよらないバランスを大切にしてきました。技術者は40%、あとは営業やマーケティングなどの部門で働く人たちです。また、社員の45%を女性が占めているというのも当社の特長ですね。

われわれの基本的考え方は、企業としても器を大きくしていかなければなりませんが、それと一緒に個々の人たちも自分の器を大きくしていってほしいということなんです。企業の夢と個人の夢がオーバーラップしていくことが大切だと思うんですよ。

そのためには教育が重要ですね。OJTや資格制度、海外研修や語学研修はもちろんなんですが、その他に社員ひとりひとりが経営に参加しているという意識を育てるためのシステムづくりもしています。たとえば、自分が提案し、その提案に対する討論が自由にできる制度などをいくつか設けているんです。これらの機会を通じて、われわれ自身が新しい企業神話をいま創りつつあるんだ、という意識を社員ひとりひとりに持ってもらえればというのが、私の願いです。」

美光亭志寿天夢曰く、あっ！と驚かせる快感を味わいたい。

フレキシブルなフィールド、高いレベルの技術力。ふたつの条件が満たされているから、ユニークな発想が活かされるのだ。着想と技術の二人三脚で、これもやりたい、あれも作ってみたいという夢が、実現できる。その発表は、コンピュータ社会という舞台。観客は常に新しいものを期待しているユーザーだ。今、出囃子が鳴り、舞台へのぼる時がきた。





●美光システムワークス。3つのユニークな特徴

その1：創立者は、フリーで活躍していたエンジニア
フリーでシステム設計などを手がけるには、しっかりしたノウハウやユーザーからの厚い信頼がなくてはなりません。それがハードウェア資源を最大限に活かした高性能システム開発の基盤となっています。

その2：専門の営業部を設けてある
ニーズとダイレクトで接することができ、そこから活気が生まれ、開発意欲にフィードバックされて、より活性化をはかります。

その3：新会社ならではの、イノベーショナルな体勢
〝変化を友として、自らも変身し、共に栄えよう〟のモットーに、ソフトウェア開発に着手。自由なフィールドと技術力の高さの相乗効果で、次世代コンピュータ開発にも、積極的に参画しています。

※広い情報化社会に、あなた独自の着想で、波紋を投げかけてください。

■事業内容■

- ●マイクロプロセッサ応用機器用組み込みソフトウェアの開発
- ●ROM化されるソフトウェアの設計・製造、検査保守
- ●パソコン応用システム用ソフトウェア
- ●ハード、ソフトを含めたシステム設計開発
- ●拡張保守

■職種■

- ●SE・プログラマー
- ●マイコン応用機器ソフト開発技術者
- ●システム営業

※未経験者も可

学生の方の資料請求歓迎

■会社概要■

設立／昭和62年4月21日　**資本金**／1000万円
代表者／山本　侑　**社員数**／8名
年商／1億6000万円（2年度実績）
事業内容／マイクロプロセッサ応用機器用組み込みソフトウェアの開発、ROM化されるソフトウェアの設計・製造・検査保守、パーソナルコンピュータ応用システム用ソフトウェア、ハードウェア・ソフトウェアを含めたシステム設計開発、拡張保守
開発実績／●FDデュプリケータ　●FDエバリュエータ　●プリント基板配線編集システム　●FDプロテクションソフト　●トラックポジション測定装置

■募集要項■

職種　●SE・プログラマー
●マイコン応用機器ソフト開発技術者
●システム営業
資格　高卒以上、30歳位迄　経験者／35歳迄
給与　経験・年齢を考慮の上、優遇します。固定給＋歩合給制　大卒初任給／19万円
待遇　昇給年1回、賞与年2回（H2年実績5ヵ月）、交通費支給、各種社会保険完備、退職金・財形貯蓄制度有、諸手当（営業・職能・家族・役職）有、社内旅行（年2回）、クラブ活動（ゴルフ、野球etc.）
勤務地　東京　中央区日本橋（本社）
勤務時間　9：15～17：30
※フレックスタイム制
休日休暇　日曜、祝日、第1・3土曜、夏季、年末年始、有給、他
応募　電話連絡の上、履歴書（写真貼付）を持参ください。
交通　地下鉄日比谷線・都営浅草線人形町駅下車徒歩3分

有限会社 美光システムワークス
〒103 東京都中央区日本橋富沢町4-6 糸重富沢町ビル4階
美光システムワークス　Tel03-3663-7822㈹　担当／山本

NEC PC-9800シリーズ MS-DOS Ver.5.0コマンド一覧

（○内部コマンド／●外部コマンド／◎拡張セットに含まれる外部コマンド）

●ADDDRV　キャラクタ系デバイスドライバの追加登録

書式：ADDDRV 〈参照ファイル名〉

●APPEND　データファイルの検索パスの設定／解除／表示

書式：APPEND [〈パス名〉;[〈パス名〉…;]] [〈オプション〉]
　　APPEND ;

〈オプション〉
- /X:ON　データファイル検索パスを利用するコマンドを追加する
- /X:OFF　/X:ONオプションを取り消す
- /PATH:ON　パス名が指定されていても，コマンド検索パス内でファイルを検索する
- /PATH:OFF　/PATH:ONオプションを取り消す
- /E　環境変数APPENDにデータファイル検索パスを記録する
- /R　メモリを解放する

●ASSIGN　論理装置を物理装置に割り当てる

書式：ASSIGN [〈論理装置〉=〈物理装置〉 [⋯]]

〈論理装置〉
- CI　コンソール入力
- CO　コンソール出力
- PR　プリンタ出力
- AI　補助入力
- AO　補助出力

〈物理装置〉
- KD(KEYBOARD)　キーボード
- SN(SCREEN)　スクリーン
- P0(RS232C-0)　標準RS-232Cインターフェース
- P1(RS232C-1)　RS-232Cインターフェース拡張ポート1
- P2(RS232C-2)　RS-232Cインターフェース拡張ポート2
- PR(PRINTER)　プリンタ
- DY(DUMMY)　ダミー

●ATTRIB　ファイルの属性の設定／解除／表示

書式：ATTRIB [〈属性〉] 〈ファイル名〉 [〈オプション〉]

〈属性〉
- +R　指定したファイルの属性をリードオンリーにする
- -R　リードオンリー属性を解除する
- +A　指定したファイルの属性をアーカイブにする
- -A　アーカイブ属性を解除する
- +S　指定したファイルの属性をシステムにする
- -S　システム属性を解除する
- +H　指定したファイルの属性を不可視にする
- -H　不可視属性を解除する

〈オプション〉
- /S　サブディレクトリも含めて，指定したパス以下の全ファイルの属性を変更する

●BATKEY　指定したメッセージ表示後，キー入力待ち(バッチファイル)

書式：BATKEY [0 | 1 | 2] [〈メッセージ〉]

〈スイッチ〉
- 0　数字の入力を待ち，エラーレベルに入力された数字を返す
- 1　[Y]また[N]の入力を待ち，エラーレベルに[Y]は0，[N]は255を返す
- 2　エラーレベルに入力されたキーのASCIIコードを返す

●BACKUP　ファイル／ディスクのバックアップ

書式：BACKUP 〈転送元ドライブ名:〉[〈ファイル名〉] 〈転送先ドライブ名〉
　　[〈オプション〉]

〈オプション〉
- /S　サブディレクトリごとバックアップする
- /M　最後のバックアップ以降更新されたファイルのみバックアップする
- /A　受け側に記録されているファイルを消さずにバックアップする
- /F:〈size〉　受け側のディスクをフォーマットしたあとでバックアップする
　〈size〉:1(1MB)／720(720KB)／640(640KB)
- /D:〈date〉　指定した日付以降に変更されたファイルだけをバックアップ
- /T:〈time〉　指定した時刻以降に変更されたファイルだけをバックアップ
- /L:〈name〉　指定したファイル名でログを残す

○BREAK　[CTRL]+[C]または[STOP]で中断できるかの設定

書式：BREAK [ON | OFF]
- ON　すべてのMS-DOSの機能を実行するたびにチェックする
- OFF　コンソール入力，プリンタ出力時のみチェックする

○CHDIR(CD)　カレントディレクトリの移動／表示

書式：CHDIR 〈パス名〉
　　CHDIR ..

●CHGEV　DOSシェルからのアプリケーションの起動(単独実行不可)

●CHKDSK　ファイル／ディスクの検査／状況報告

書式：CHKDSK 〈ファイル名〉 [〈オプション〉]

〈オプション〉
- /V　ディスクの中のすべてのファイルを表示する
- /F　ディスクに論理エラーがある場合は修復する

●CHKENV　アプリケーションの実行環境の調査

書式：CHKENV 〈ファイル名〉

●CHKFIL　SETUPコマンドで使われるプログラム(単独実行不可)

○CLS　テキスト画面の消去

書式：CLS

●COMMAND　コマンドプロセッサ

書式：COMMAND [〈再ロードするシェルのファイル名〉] [〈オプション〉]

〈オプション〉
- /E:〈size〉　環境変数が使うメモリのサイズを指定する(既定値は256バイト)
- /P　上位レベルへの抜け出しを禁止する
- /C〈コマンド〉　指定したコマンド実行後，COMMAND.COMを終了する
- /MSG　エラーメッセージをメモリに読み込む

○COPY　ファイルの複写

書式：COPY [〈ファイル名1〉][オプション1] [〈ファイル名2〉][オプション2]
　　COPY 〈ファイル名1〉+[〈ファイル名2〉…][オプション1]〈ファイル名3〉
　　[オプション2]

〈オプション1〉
- /A　ファイル1をテキストファイルとして扱う
- /B　EOFを無視してファイル全体を複写する

〈オプション2〉
- /A　受け側ファイルの最後にEOFコードを書き込む
- /B　受け側ファイルの最後にEOFコードを書き込まない
- /V　複写終了後，正しく書き込まれたかチェックする

●COPYA　RS-232Cを使ったファイルの複写

書式：COPYA 〈オプション〉 〈ファイル名〉 AUX
　　COPYA 〈オプション〉 AUX 〈ファイル名〉

〈オプション〉
- /B　ファイルをバイナリモードで送受信する

○CTTY　標準入出力デバイスの変更

書式：CTTY 〈デバイス名〉

●CUSTOM　CONFIG.SYSファイルとADDDRV用定義ファイルの作成／更新

書式：CUSTOM 〈ADDDRV用定義ファイル名〉

○DATE　システム時刻の表示／設定

書式：DATE [〈日付〉]

◎DEBUG　デバッガ

書式：DEBUG 〈デバッグファイル名〉 [〈引数リスト〉]

○DEL(ERASE)　ファイルの削除

書式：DEL 〈ファイル名〉 [〈オプション〉]

〈オプション〉
- /P　削除前に確認を求める

●DELDRV　ADDDRVで登録したデバイスドライバの削除

書式：DELDRV

●DICM　辞書ファイルの保守

書式：DICM 〈辞書ファイル名〉

○DIR　ディレクトリ内容の表示

書式：DIR [〈ファイル名〉] [〈オプション〉]

〈オプション〉
- /P　1画面分表示ごとに停止する
- /W　ファイル名だけを，1行に5個ずつ表示する
- /A:〈属性〉　指定した属性のファイルだけ表示する
- /O:〈ソート順〉　ファイルを指定したソート順で表示する
- /S　サブディレクトリの中のファイルも表示する
- /B　ファイル名だけ表示する
- /L　小文字で表示する

●DISKCOPY　フロッピーディスクのバックアップ／照合

書式：DISKCOPY [〈複写元ドライブ名〉] [〈複写先ドライブ名〉] [〈オプション〉]

〈オプション〉
- /V　複写しないでディスクの比較照合だけを行う
- /P　キー入力を促すメッセージを表示しない
- /Q　複写中に，データを照合しない

●DOSKEY　コマンド入力ユーティリティ

書式：DOSKEY 〈オプション〉

〈オプション〉
- /REINSTALL　DOSKEYが組み込まれている状態で，さらに組み込む
- /BUFSIZE=〈size〉　マクロなどを記録するバッファサイズを指定する
- /MACROS(/M)　定義済みのマクロを表示する
- /HISTORY(/H)　コマンドのヒストリーを表示する
- /INSERT　挿入モードで起動する
- /OVERSTRIKE　上書きモードで起動する
- /R　メモリを解放する

●DOSSHELL　DOSシェル

書式：DOSSHELL 〈オプション〉

〈オプション〉
- /T　テキストモードで起動する
- /G　グラフィックモードで起動する
- /B　モノクロモードで起動する

●DOSSWAP(単独実行不可)

●DUMP　ファイル内容の16進数表示

書式：DUMP [〈ファイル名〉 [〈開始アドレス〉 [〈終了アドレス〉]]] [〈オプション〉]

〈オプション〉
- /D　アドレスを10進数表現で指定する

●EDLIN　ラインエディタ

書式：EDLIN 〈ファイル名〉 [〈オプション〉]

〈オプション〉
- /B　EOFコードをファイルの終端と見なさない

●EMM386　EMM386.EXEによって確保されたEMSメモリの制御

書式：EMM386 [ON | OFF | AUTO]
- ON　EMM386.EXEを有効にする
- OFF　EMM386.EXEを一時的に無効にする
- AUTO　EMM386.EXEを自動モードにする

◎EXE2BIN　拡張子に.EXEをもった実行ファイルをバイナリ形式に変換

書式：EXE2BIN 〈実行ファイル名〉 〈バイナリファイル名〉

○EXIT　子プロセスとして起動されたCOMMAND.COMを終了し，親プロセスに戻る

書式：EXIT

●FASTOPEN　ファイルアクセスの高速化

書式：FASTOPEN 〈ドライブ名〉=〈ディレクトリエントリ数〉 [〈オプション〉]

〈オプション〉
- /E　キャッシュとして利用するメモリをEMS上に確保する

●FC　ファイルの比較

書式：FC [〈オプション〉] 〈ファイル名1〉 〈ファイル名2〉

〈オプション〉
- /A　アスキー形式で比較する(出力は短縮形)
- /B　バイナリ形式で比較する(バイナリファイル比較時のデフォルト)
- /C　大文字と小文字を区別しないで比較する
- /L　アスキー形式で比較する(テキストファイル比較時のデフォルト)
- /N　行番号を表示する(アスキー形式での比較時)
- /T　タブを空白に展開する
- /W　タブや空白は違いとして区別しない
- /〈n〉　〈n〉で指定した行数だけ一致していると，一致した部分をみなす
- /LB〈n〉　内部のラインバッファを〈n〉行に設定する

●FILECONV　MS-DOSとN88-BASIC間のファイル変換

書式：FILECONV

●FIND　ファイルの中から指定文字の検索

書式：FIND [〈オプション〉] 〈文字列〉 〈ファイル名〉 [⋯]

〈オプション〉
- /V　指定した文字列を含まないすべての行を表示する
- /C　指定した文字列の含まれる行の数だけ表示する
- /N　相対的な行番号を左に表示する
- /I　大文字と小文字の違いを無視して検索する

●FORMAT　フロッピーディスク／ハードディスクの初期化

書式：FORMAT [〈ドライブ名〉] [〈オプション〉]

〈オプション〉
- /S　初期化後にシステムファイルも複写する
- /V　ボリュームラベルを付ける
- /6　640KBのフォーマットをする
- /9　720KBのフォーマットをする
- /M　1MBのフォーマットをする
- /H　ハードディスクまたは光ディスクをフォーマットする
- /E　より簡単にハードディスクまたは光ディスクをフォーマットする
- /P　プロンプトを省略する
- /B　システム転送領域を用意する(約124KB)
- /Q　クイックフォーマットをする
- /U　無条件で物理フォーマットをする
- /?　ヘルプを表示する

〈未公開のオプション〉
- /5　IBM互換の512KB/セクタでフォーマットをする
- /C　/Qオプションと同じ

●HDFORMAT　ハードディスクのフォーマット(単独実行不可)

●HDUTL　ハードディスクのメインテナンス
書式：HDUTL

●JOIN　ドライブをディレクトリに結合
書式：JOIN [〈ドライブ名〉〈パス名〉]
　　JOIN 〈ドライブ名〉/D
〈オプション〉
/D　JOINで結合したドライブを解除する

●KEY　ファンクションキー／カーソルキーへの機能の割り当て／取り消し
書式：KEY 〈ファイル名〉[〈オプション〉]
〈オプション〉
/S　[CTRL]+[ファンクションキー]を有効にする
/N　[CTRL]+[ファンクションキー]を無効にする

●LABEL　ボリュームラベルの作成／変更／削除
書式：LABEL [〈ドライブ名:〉][〈ボリュームラベル名〉]

◎LIB　ライブラリファイルの作成／管理
書式：LIB [〈ライブラリファイル名〉[オプション]〈コマンド〉… [,〈リストファイル名〉,〈出力ファイル名〉]]

◎LINK　オブジェクトモジュールのリンク
書式：LINK [[〈オプション〉]〈オブジェクトファイル名〉…,[〈出力ファイル名〉],[〈マップファイル名〉],[〈ライブラリファイル名〉…]]

○LOADHIGH(LH)　UMBメモリへのプログラムのロード
書式：LOADHIGH 〈ファイル名〉[〈パラメータ〉]

◎MAKE　プログラム開発工程を管理するユーティリティ
書式：MAKE 〈メイクファイル名〉

●MAOIX　SETUPユーティリティからアプリケーションの登録補助(単独実行不可)

◎MAPSYM　シンボルファイルの作成
書式：MAPSYM [〈オプション〉]〈シンボルマップファイル名〉

●MEM　システムメモリの表示
書式：MEM [〈オプション〉]
〈オプション〉
/P(PROGRAM)　メモリ中のプログラムのロード状況を表示する
/D(DEBUG)　/Pオプションよりさらに詳しく表示する
/C(CLASSIFY)　プログラムのメモリ使用方法を分類して表示する

●MENUCONV　MENU.COMの設定をDOSシェル用に変換
書式：MENUCONV 〈ファイル名１〉〈ファイル名２〉

●MIRROR　ファイル復旧情報の保存
書式：MIRROR [〈オプション〉]
〈オプション〉
/T〈ドライブ名〉[-〈size〉]　ファイル削除監視プログラムを常駐する
/I　ディスクの最新情報のみ保存する
/PARTN　固定ディスクの分割情報をフロッピーディスクに保存する
/R　/Tで常駐したプログラムを終了する

○MKDIR(MD)　新しいディレクトリの作成
書式：MKDIR 〈パス名〉

●MORE　標準入力からのデータを１画面分ずつ標準出力に表示
書式：MORE

●MOUSE　DOSシェル用常駐型マウスドライバ
書式：MOUSE [〈オプション〉]
〈オプション〉
/S〈n〉　マウスの縦／横両方向の感度を指定する(0～100)
/H〈n〉　マウスの横方向の感度を指定する(0～100)
/V〈n〉　マウスの縦方向の感度を指定する(0～100)
/N〈n〉　マウスカーソルの反応の遅さを指定する(0～255)
/P〈n〉　マウスの移動時の加速度の加わり方を指定する(1～4)
/I〈n〉　割り込みベクタを指定する(3～6)
/M〈n〉　マウスカーソルのマスクを許可する(0～1)
/R　マウスドライバを外す

●MSASSIGN　ASSIGNの補助プログラム(単独実行不可)

●NECAIKEY　日本語入力で使用するファンクションキー割り付けの変更
書式：NECAIKEY [日本語入力キーファイル名]

●PATCH　プログラム修正ユーティリティ
書式：PATCH

○PATH　外部コマンドを探すディレクトリの設定／表示
書式：PATH [〈パス名〉][;〈パス名〉][;…;〈パス名〉]

●PRINT　テキストファイルの印刷
書式：PRINT [[〈オプション１〉]〈ファイル名〉][〈オプション２〉]
〈オプション１〉
/D:〈デバイス名〉　デバイスを指定する
/B:〈size〉　内部バッファ数を指定する
/U:〈n〉　プリント待ち時間を指定する
/M:〈n〉　プリント側の使用時間を指定する
/S:〈n〉　タイムスライス値を指定する
/Q:〈n〉　印刷待ち行列のファイル数を指定する
〈オプション２〉
/T　プリント待ち行列中のファイルを削除する
/C　取り消しモード
/P　プリントモード
/R　常駐を取り消す

○PROMPT　コマンドプロンプトの変更
書式：PROMPT 〈プロンプトテキスト〉

●RECOVER　ファイルやディスクの修復
書式：RECOVER 〈ファイル名〉

○RENAME(REN)　ファイル名の変更
書式：REN 〈変更前ファイル名〉〈変更後ファイル名〉

●RENDIR　ディレクトリ名の変更
書式：RENDIR 〈変更前ディレクトリ名〉〈変更後ディレクトリ名〉

●REPLACE　ファイルの更新
書式：REPLACE 〈ファイル名〉[〈パス名〉][〈オプション〉]
〈オプション〉
/A　追加(上書きされない)
/U　日付比較日時の新しいものだけ置き換える
/P　置き換える前に,確認を求める
/R　読み取り専用ファイルの置き換える
/S　置き換え先のサブディレクトリまで検索して置き換える
/W　一時停止する

●RESTORE　バックアップファイルの復元
書式：RESTORE 〈バックアップディスクのドライブ名〉:〈復元するファイル名〉[〈オプション〉]
〈オプション〉
/S　サブディレクトリも復元する
/P　隠しファイルや読み取り専用ファイルの場合,確認を求める
/B:〈日付〉　指定された日付より前に変更されたファイルだけ復元する
/A:〈日付〉　指定された日付よりあとに変更されたファイルだけ復元する
/E:〈時刻〉　指定された時刻より前に変更されたファイルだけ復元する
/L:〈時刻〉　指定された時刻よりあとに変更されたファイルだけ復元する
/M　最後のバックアップ以降に変更のあったファイルのみ復元する
/N　復元先に存在しないファイルだけ復元する
/D　復元は行わず,指定されたファイルのリストを表示する

○RMDIR(RD)　サブディレクトリの削除
書式：RMDIR 〈パス名〉

●SEDIT　スクリーンエディタ
書式：SEDIT 〈ファイル名〉

●SELKKC　DOS 5.0対応の日本語入力FEPの切り替え
書式：SELKKC

○SET　環境変数の設定／表示／削除
書式：SET [〈環境変数〉=〈文字列〉]

●SETUP　アプリケーションプログラムの登録
書式：SETUP

●SETUP2　アプリケーションプログラム登録の補助(単独実行不可)

●SETVER　バージョン番号の設定
書式：SETVER 〈ファイル名〉〈バージョン番号〉
　　SETVER 〈ファイル名〉[〈オプション〉]
〈オプション〉
/DELETE(/D)　指定したファイルをバージョンテーブルから削除する
/QUIET(/Q)　/Dと同時に指定し,バージョンテーブルから削除するときメッセージを表示しない

●SHARE　ネットワーク上でファイルの共有とロック
書式：SHARE [〈オプション〉]
〈オプション〉
/F:〈n〉　ファイル共有情報用のメモリ容量を指定する(省略時は2048バイト)
/L:〈n〉　同時にロック可能な数を指定する(省略時は20)
/R　SHAREコマンドをメモリから取り除く

●SORT　標準入力からのデータを並べ替え,標準出力に表示
書式：SORT [オプション]
〈オプション〉
/R　逆順のソートつまりＺからＡの順に並べ替える
/+〈n〉　n桁目の文字を並べ替えの対象とする(省略時は１桁目)

●SPEED　RS-232Cのパラメータ設定
書式：SPEED [〈ポート番号〉[〈パラメータ〉]]

●SUBST　パス名をドライブ名で置き換え
書式：SUBST 〈ドライブ名〉[〈パス名〉][〈オプション〉]
〈オプション〉
/D　パス名の置き換えを解除する

●SWITCH　メモリスイッチの表示／変更
書式：SWITCH [〈パラメータ〉]

◎SYMDEB　シンボリックデバッガ
書式：SYMDEB [〈シンボルファイル名〉][〈デバッグファイル名〉[〈引数リスト〉]]

●SYS　システムファイルの複写
書式：SYS 〈複写元ドライブ名〉〈複写先ドライブ名〉[〈オプション〉]
〈オプション〉
/O　COMMAND.COMは複写しない

○TIME　システム時刻の表示／設定
書式：TIME [〈時刻文字列〉]

●TREE　パスとファイルの一覧表示
書式：TREE [〈パス名〉][〈オプション〉]
〈オプション〉
/A　グラフィック文字を使用しない
/F　各ディレクトリのファイル名も表示する

○TYPE　ファイル内容の表示
書式：TYPE 〈ファイル名〉

●UNDELETE　誤って削除したファイルの復旧
書式：UNDELETE [〈ファイル名〉][〈オプション〉]
〈オプション〉
/LIST　復旧できるファイルを表示する
/DT　削除追跡ファイルだけを元に復旧する
/DOS　ディレクトリだけ調べて復旧する
/ALL　自動的に復旧する

●UNFORMAT　誤ってフォーマットしたディスクの復旧
書式：UNFORMAT [〈ドライブ名〉][〈オプション〉]
〈オプション〉
/J　MIRRORコマンドで作成したファイルと,FORMATコマンドで作成したMIRRORイメージを比較する
/L　発見したディレクトリとファイルを表示,/PARTNと一緒に指定した場合にはパーティションを表示する
/P　実行経過をプリンタに出力する
/TEST　MIRRORコマンドで作成したファイルを使用しないで復旧できるかチェックする
/U　MIRRORコマンドで作成したファイルを使用しないで復旧を試みる
/PATRN　パーティションを復旧する

●USKCGM　ユーザー定義文字の保守
書式：USKCGM [〈ファイル名〉][〈オプション〉]
〈オプション〉
/P　プリンタに外字フォントを送る

○VER　MS-DOSバージョン番号の表示
書式：VER

○VERIFY　ディスク書き込み時,照合するかの設定／表示
書式：VERIFY [ON | OFF]
ON　照合する
OFF　照合しない

○VOL　ボリュームラベルの表示
書式：VOL [〈ドライブ名〉:]

●XCOPY　高速大量複写コマンド
書式：XCOPY 〈複写元ファイル名〉〈複写先ファイル名〉[〈オプション〉]
〈オプション〉
/A　保存(アーカイブ)属性がついたファイルだけ複写する
/D:〈日付〉　指定した日付以降に変更されたファイルを複写する
/E　/Sとともに使用し,空のディレクトリも複写する
/M　/Aと同じ,元のファイルの保存属性を解除する
/P　複写するごとに,確認を求める
/S　サブディレクトリの内容も複写する

マスコミに取り上げられる SEでありたい人しか要らない

というこことで、マスコミによく取り上げられるＳＥのライフスタイルを分析してみました。
下記の３つのうち、どれかに自分がピッタリとくると思う方、奮ってご応募下さい。
ただし、２つは全くマスコミに取り上げられたことのないＳＥのライフスタイルです。
ピタリ的中でのご応募の方は、当社の看板ＳＥとして、
華々しくデビューを飾れるよう、育成、援助します。

	特長	仕事	機種	食	遊&休	情報	住	異性・結婚
a	・肉体的、精神的に自己研鑽を怠らない ・自己のアイデンティティーを持っている ・判断基準が確立している ・関心領域が広い	・自分にあった仕事をするのが一番と考え、仕事をエンジョイする ・失敗してもくじけず糧として頑張る根性ＳＥ ・社会に自分の名前を売れる仕事に関心が高い	・自分流ハードの使い方スタイルを持っている ・自分に合うもの合わないものの判断ができる ・クオリティー重視 ・コーディネート能力抜群	・食べることに貪欲 ・健康のために食を重視――栄養のバランスに気をつける ・行きつけの喫茶店がある	・流行に流されず自分流の遊びを追求 ・アウトドア志向、感性追求型 ・気の合う仲間がたくさんいる ・リラックスタイムも充実	・情報収集に超熱心であり、気になる情報はストック ・膨大な情報の整理・判断ができる ・ヒト情報重視、ネットワークの中心人物	・コーディネート感覚に富む ・素材を見る目とコスト感覚が鋭い ・自宅でよくホームパーティーっぽいことを催す	・自分にないものを持っていたり、何か一つのことに精通している異性 ・「空気みたいな」異性 ・結婚には多少抵抗を感じながらも肯定的 ・恋愛に関して若干優柔不断なところあり
b	・新しいものへの関心が異常に高い ・超積極的 ・熱しやすく冷めやすい	・人との打合せが大好きなおしゃべりＳＥ ・仕事は浅く、広く ・仕事だけの生活はイヤ、休みは何があっても取る ・自分だけに責任の掛かる仕事は敬遠	・ハードに対する思い入れが非常に高い ・ハードの流行を追うが、その中に自分らしさも出す ・どんなジョブでもまず、ハードから入る	・体型の維持を一番に考えて食事をする ・流行の店は行かなくても、いつか行こうと覚えている ・３度の食事以外はタブー視する	・話題のイベントはチェックし、必ず行く ・気の合ったグループで行動 ・遊ぶ、静かにしている、メリハリのある生活	・自動車、ＡＶ機器の情報に強い ・情報の鮮度は重視、「それはもう死語！」でよく人の話を切り返す ・広く、浅く	・ＡＶビンボウ、でも頻繁には使用してない ・電話命、マメで長電話 ・自分の部屋はプランニング場所 ・日曜の不動産チラシに乗っているモデルルーム的なレイアウト	・特別な関係の異性が２～３人なんてアタリマエ ・異性をうまく使い分ける、時間をバッティングなんてヘマは絶対に起こさない ・熱しやすく冷めやすい
c	・ワンパターンな生活の繰り返し ・変化が嫌い ・新しいものに無関心 ・面倒くさがりや ・よく落ち込むが、すぐに忘れる	・ルーティンデベロップ命 ・仕事の変化は絶対悪と考える ・定時の終了５分前には帰る準備でめっぽう忙しそうにする	・いつまでも、どこまでも同じ機種にこだわる ・ジョブ別のハードなど考えたこともない ・コーディネートという概念は生活習慣の中にない	・おなかが鳴ったらたべる ・栄養のバランスなど考えない ・一応体型は気にするが欲求には勝てず、アートさんのお世話になる可能性も高い	・誰かが誘ってくれるのを待っている ・家でのゴロゴロが一番の楽しみ ・きまった友達と過ごす時間が長い。しかし基本的にその友達のことは内心嫌いである	・情報武装している（オタクっぽい） ・耳年増 ・定期購読している雑誌をむさぼり読む ・全く知らない分野もある	・自分創造の城 ・主張のないインテリア ・自分さえよければ掃除はしない ・コップや食器は、もらい物か何かの景品	・特別な関係の異性いない歴10年はアタリマエ（たぶんです） ・理想は社長の給料よりも高い ・結婚願望は強烈……ただし、だれでもいいのではない、もちろん玉の輿ｏｒ逆玉

会社プロフィール

設立●昭和59年4月2日 代表者●代表取締役 鈴木和芳 資本金●990万円 従業員数●144名(男:105名、女:39名) 平均年齢●25.5歳 売上高●8億8000万円(平成3年度実績) 事業内容●①CDIソフトの開発 ②FA&ロボットなどの自動制御 ③データ通信&自動車電話などの通信制御 ④TV局向け&医療向けの映像制御・画像処理 ⑤オンライン端末&無線端末などのマイクロソフト制御 上記分野を中心としたソフトウェア・ハードウェアの企画、開発、販売、コンサルティング 主要機種●HP64000、HP9000LAN、PC9801、FMR、Macintosh他多数 主要言語●C、アセンブラ、他 主要取引先●日本電気㈱、日本電気無線電子㈱、日本無線㈱、富士通機電㈱、松下システムエンジニアリング㈱、松下通信工業㈱、京セラ㈱、JRCエンジニアリング㈱、東芝エンジニアリング㈱、昭和電子㈱

募集要項

職種●コンピュータエンジニア ※今春・来春新卒者のお問い合せ歓迎 ※学生アルバイト同時募集 資格●学歴不問 26歳位迄 給与●月給 固定給制19万円以上 ※経験・年齢・前職を考慮の上、相談により決定します。26歳支給例／31万6500円 アルバイト／時給1000円以上(時間相談に応じます。) 待遇●昇給年1回(3年度実績12%)、賞与年2回(2年度実績7ヵ月)、通勤交通費(2万4000円迄支給)、各種社会保険完備、住宅・家族・精勤・時間外・役職手当、退職金制度、財形貯蓄制度、社内旅行(グアム)、クラブ活動(野球部、テニス部、バスケット部、ウインドサーフィン部 他多数クラブ ※会社から援助金支給)、ラップトップ・レンタル制度(長期間無料) 勤務地●本社(相模原) 勤務時間●フレックスタイム制(コアタイム10:30～15:30) ※タイムカードはありません。1日標準労働時間8時間 休日休暇●完全週休2日制(土・日)、祝日、夏期・年末年始・GW・有給(初年度8日から)・慶弔休暇 ※年間125日＋有給 応募●電話連絡の上、履歴書(写真貼付)・職務経歴書(職歴のない方は自己PR書)を持参ください。郵送も可。 ※FAX応募も歓迎します。 ※応募の秘密厳守します。 ※入社日は相談に応じます。 ※新卒者の資料請求歓迎します。 ※大学生の会社訪問歓迎します。 交通●ＪＲ横浜線相模原駅徒歩5分

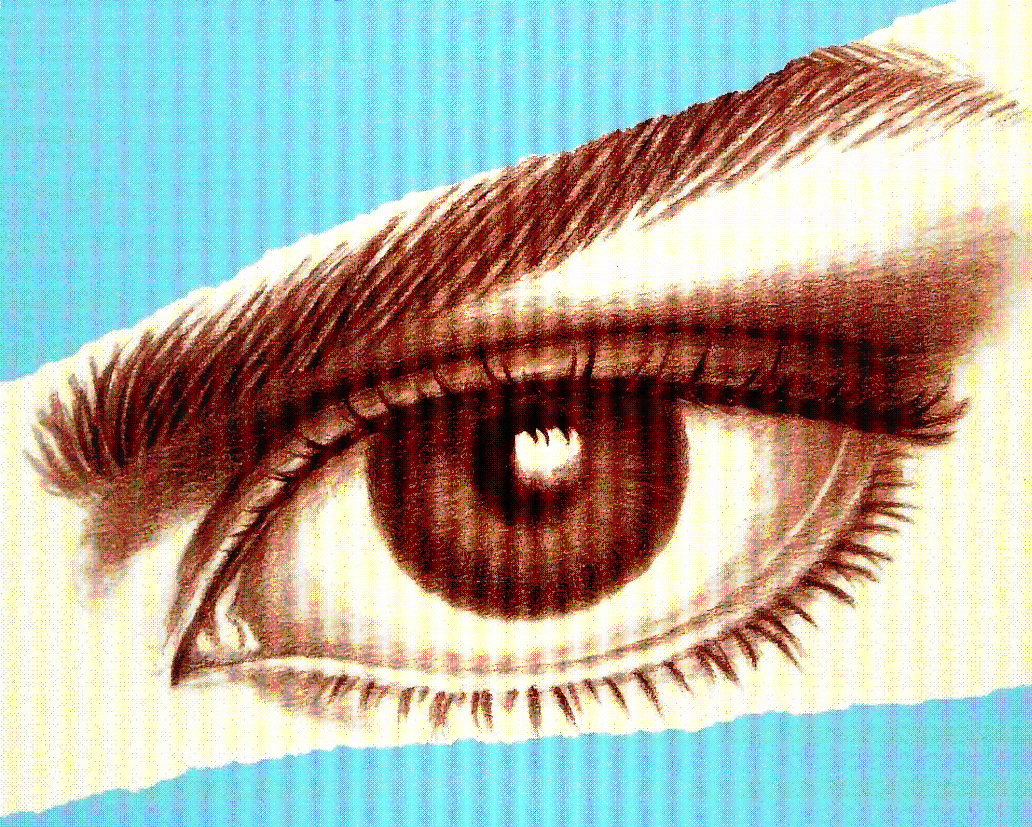
採用関係連絡先

〒229 神奈川県相模原市相模原4-3-14相模原第一生命ビル「株式会社ソフトビジョン 人事部人事課」
フリーダイヤル0120-211744 採用担当／海老沢、斉藤
FAX 0427-51-3333

株式会社 ソフトビジョン

BORLAND

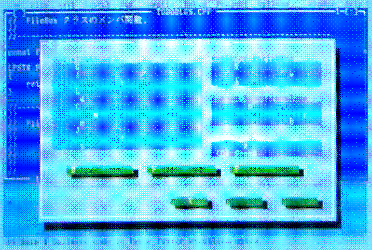
The Leader in Object-Oriented Programming



私の方が速い。

BORLAND C++ & APPLICATION FRAMEWORKS 3.0（以下BC++ & AFX 3.0）は、プロ向けの超高速最適化コンパイラです。ANSI CとAT&T C++ 2.1をサポートし、テンプレートも使えます。強力な最適化機能によりコードを速く、小さくすることも可能になりました。豊富なツール類がDOSとWindowsの大規模なアプリケーション開発を強力に支援します。

■強力な最適化機能搭載の新・C&C++ コンパイラ



●BC++のオプティマイズオプション画面

広範囲にわたる革新的な最適化が最小のコードと最高の実行速度を提供します。

■プログラマーをサポートするさまざまな機能

BC++ & AFX 3.0には、最適化後のコードのデバッグも可能なTurbo Debugger、オブジェクト指向に拡張されたTurbo Assembler、Windowsリソースの編集ができるResource Workshop、Windows対応したプログラムのボトルネックを調べるTurbo Profiler、Windowsメッセージを監視するWinsight、DOSの標準入出力をWindows上でエミュレートするEasywin、Windows上の新統合環境TC++ for Windows 3.0など、便利なツール類や、従来は別売だった標準ライブラリとAFXのソースコードまでもが含まれています。もちろんコンパイルスピードも飛躍的に向上しており、BC++ & AFX 3.0だけで、DOSとWindows上のあらゆるアプリケーションの開発を快適に行なうことができます。

■製品内容…マニュアル13冊・ディスク13枚

■動作環境…●使用機種：NEC PC-9800シリーズ（ハイレゾモード対応、XAを除く）、EPSON 286/386シリーズ（CPU 80286以上のマシン）●メモリ：要プロテクトメモリ・1MB以上*注（推奨5MB以上）●OS：MS-DOS Ver.3.3以上、MS-Windows Ver.3.0以上（Windowsアプリケーション作成時のみ）●周辺機器：要ハードディスクドライブ・15MB以上の空き領域（推奨容量50MB）、バスマウス対応

■価格……………………………¥98,000（税別）

*注：今バージョンからプロテクトモードのコンパイラのみとなります。



BORLAND® C++
& APPLICATION FRAMEWORKS 3.0

ボーランド株式会社

〒151 東京都渋谷区笹塚1-64-8 笹塚サウスビル TEL.03-5350-9370㈹

※詳しくは、カタログをご請求ください。

T1017777050984　〈資料請求番号P04〉　雑誌コード17777-05